Interactive Refinement of Combinatorial Test Plans

Itai Segall
IBM, Haifa Research Lab
Haifa University Campus
Haifa, 31905, Israel
itais@il.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 planning technique that reveals faulty feature interactions in a given system. The test space is modeled by a set of parameters, their respective values, and restrictions on the value combinations. A subset of the test space is then automatically constructed so that it covers all valid value combinations of every t parameters, where t is a user input.**

When applying CTD to real-life testing problems, it can often occur that the result of CTD cannot be used as is, and manual modifications to the tests are performed. One example is very limited resources that significantly reduce the number of tests that can be used. Another example is complex restrictions that are not captured in the model of the test space. The main concern is that manually modifying the result of CTD might potentially introduce coverage gaps that the user is unaware of. In this paper we present a tool that supports interactive modification of a combinatorial test plan, both manually and with tool assistance. For each modification, the tool displays the new coverage gaps that will be introduced, and enables the user to take educated decisions on what to include in the final set of tests.

I. Introduction

As software systems become increasingly more complex, verifying their correctness is even more challenging. The introduction of service-oriented architectures (SOA) contributes to the growing trend of highly configurable systems, in which many optional features coexist and might unintentionally interact with each other in a faulty way. Verification approaches such as formal verification and model based testing are highly sensitive to the size and complexity of the software, and might require extremely expensive resources. Functional testing, on the other hand, is prone to omissions, as it always involves a selection of what to test from a possibly enormous space of scenarios, configurations, or conditions. It therefore requires careful consideration of what to include in the testing. Combinatorial test design (CTD) is an effective test planning technique, in which the space to be tested is modeled by a set of parameters, their respective values, and restrictions on the value combinations. A subset of the space is then automatically constructed so that it covers all valid value combinations (a.k.a interactions) of every t parameters, where t is a user input. The most common application of CTD is known as pairwise testing, in which the interaction of every pair of parameters is covered.

The reasoning behind CTD is the observation that in most cases the appearance of a bug depends on the combination of a small number of parameter values of the system under test. 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], [1]. Other experimental work has shown that typically 100% of bugs can be revealed by covering the interaction of between 4 and 6 parameters [4].

Much research has been invested in constructing effective and efficient CTD tools and algorithms [2], [5]. One example is the tool developed in our group [6]. However, our experience shows that some users still tend to manually modify test plans, even those generated by highly efficient optimization tools. One reason is testing resources that are extremely limited compared to the size of the test space. In this case even covering every single value might result in too many tests, thus some tests need to be removed. Another reason is that capturing the entire set of restrictions that correctly define the test space may sometimes be a highly complicated task. Thus, some users tend to omit the more complex restrictions from the input model, and manually modify the resulting test plan to consist only of valid tests.

The main concern is that by manual modification of CTD-generated test plans, the user might unintentionally introduce significant coverage gaps to the test plan. Since the result of CTD is optimized, each test may contain many unique combinations, thus removing or modifying even a single test might have unexpected consequences. In this paper, we present a tool that supports interactive modification of combinatorial test plans. In the tool, one can modify the test plan, either manually, or with tool assistance, while getting continuous feedback on the effect of each modification on the coverage gaps of the test plan. The tool provides an effective way of making educated decisions when one comes to modify a test plan, based on knowledge of what coverage is maintained and what coverage is omitted in the final set of tests. A pre-recorded demo of the tool is available at [3].

978-1-4673-1067-3/12/$31.00 © 2012 IEEE

II. Background

A. Combinatorial Test Design (CTD)

Combinatorial Test Design (CTD) is a well-known, powerful test planning technique, in which all value combinations of size t of the parameters of a system are tested.

The technique calls for identifying a set of parameters and their corresponding values. Optionally, restrictions can be stated on combinations of values that are not to appear in a test. We refer to these parameters, values and restrictions as a *model*. A test plan is then a set of tests, where a test is represented by a tuple that assigns a value to each parameter.

Given a model, and an integer t, CTD generates such a test plan in which for every t parameters, every valid combination of values for them appears in at least one test. A combination of values is valid if it is not excluded by any restriction. We refer to the above integer t as the *level of interaction*. In general, one could require different levels of interaction for different subsets of parameters. We generally refer to these requirements as *interaction coverage requirements*, or simply *coverage requirements*. A common special case is one in which $t = 2$, which is usually referred to as *pairwise testing*.

For example, consider testing an operating system for cellphones. A test in this setting could be a configuration of the cellphone, e.g., whether it has a GPS or not, a WiFi card or not, etc. An example of a restriction in this setting would be one that excludes phones that support 4G but do not support 3G.

B. Coverage Holes Analysis

Coverage holes analysis is a method of analyzing a test plan in light of a model. Given a test plan, a model and a level of interaction t, this analysis lists all combinations of values of size $\leq t$ that are valid in the model, yet not covered by the test plan (i.e., do not appear in any test). We refer to such combinations as *holes*.

Typically, the coverage holes analysis omits holes that are contained in larger ones. For example, if a test plan for the above example model does not contain any test in which a GPS exists, then by definition, it does not contain any test in which both GPS and WiFi exist. The coverage holes analysis will thus report only the former, but not the latter.

Similarly to the CTD algorithm, the coverage holes analysis can handle multiple coverage requirements, i.e., different levels of interaction for different subsets of the parameters.

III. Interactive Refinement of Test Plans

The interactive refinement tool (implemented as part of IBM's CTD tool [6]) allows one to interactively modify a test plan for a given model, while receiving continuous feedback on the effect of each modification, and the overall status of the current version of the test plan. The feedback is given in the form of a coverage holes analysis that shows the holes in the current version of the test plan, as well as a preview of the holes that will be introduced or covered if different operations are to be performed.

The main view of the tool is given in Figure 1. It consists of three parts, as follows. The top pane displays the current test plan, given as a list of assignments of values to parameters. The bottom pane displays the interaction coverage requirements (pairwise interaction in the example). Finally, the middle pane displays the coverage holes analysis for the current test plan, in light of the current coverage requirements. The holes in the current test plan are the ones appearing in non-bold font. The bold ones will be explained shortly.

Note that the holes view gets continuously updated as the test plan or coverage requirements change, and also acts as a preview pane before performing certain operations, as detailed below. This allows the user to consider the effects of her operations before actually performing them, as well as having the global picture of the total value of the current test plan, in terms of interaction coverage.

A. Removing Tests

Removing tests from test plans, usually due to time or resource constraints, is a common operation. Usually, the test architect manually observes the test plan, and tries to identify the test(s) that can be removed while making acceptable sacrifices in coverage.

Our tool allows making these decisions in an educated and methodological manner, by previewing the effects of the removal before actually performing it, as well as continuously presenting the global coverage of the current test plan.

Consider the example in Figure 1, in which the sixth test is selected in the test plan pane. By selecting one or more tests, the user indicates a possible intention to remove them from the test plan. Thus, the holes analysis pane immediately reflects the "damage" that will be done by doing so, by displaying and highlighting the new holes that will be introduced. For example, in Figure 1, if the sixth test will be removed from the test plan, the combination in which CDMA is true and touchscreen is false (which is currently covered) will no longer be covered. Similarly, for the GPS=true, Touchscreen=false pair. Such "potential" holes are marked bold in the holes analysis pane, as opposed to the ones in the current test plan which are non-bold [1].

B. Adding Tests

If a test architect realizes that the interaction coverage of the current test plan is unsatisfactory, or if more testing resources become available, thus allowing for more tests to be executed, she may choose to add tests to the test plan. This can be achieved either by manually specifying a test to be added, or by allowing the tool to automatically add tests that cover certain holes of importance.

[1] Bold and non-bold fonts in this paper replace red and blue colored fonts in the tool, for better readability in black and white printing

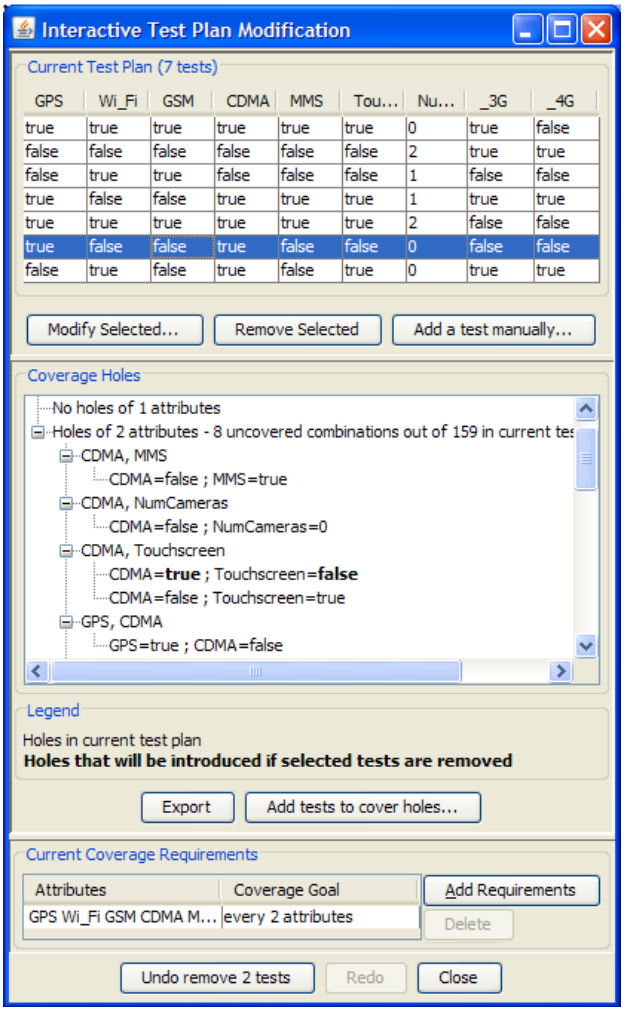

Figure 1. The main window of the interactive refinement tool, including the current test plan (top), the current coverage requirements (bottom), and the current coverage holes analysis (middle). The sixth test has been marked as a candidate for removal.

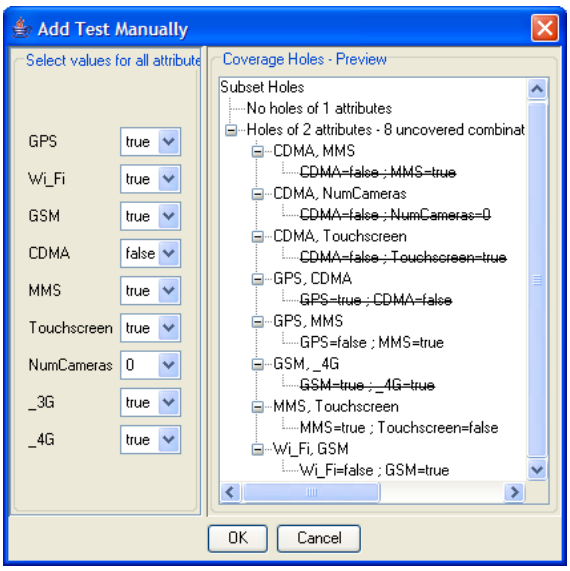

Figure 2. The dialog for adding a test manually to the test plan, showing the current test to be added, and a preview of its effect on the holes analysis.

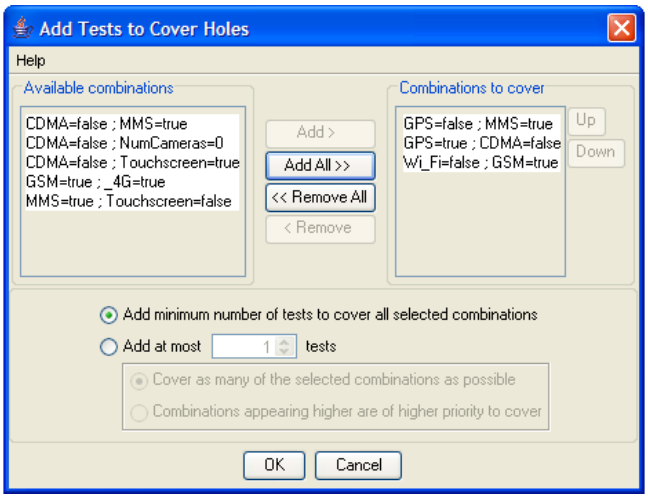

Figure 3. The dialog for adding a test automatically, by choosing, and possibly prioritizing, the holes to be covered.

1) Adding Tests Manually: Figure 2 illustrates the GUI for adding a test manually. On the left hand side, the user chooses values for the parameters, while the right hand side continuously previews the effect of this operation on the coverage holes analysis. Holes that will be covered by the new test are designated by a strikethrough font in the preview pane. In the example of Figure 2, by adding a test in which the cellphone has all features other than CDMA support and cameras, one could cover 5 out of the current 8 holes.

2) Tool-Assisted Test Addition: Figure 3 shows the GUI for tool-assisted addition of tests. In this dialog, the user chooses out of the current holes the ones she is interested in covering. This choice is usually based on an estimation of which combinations are more likely to yield defects. For example, in the figure, the user identified the combinations (GPS=false, MMS=true), (GPS=true, CDMA=false) and (WiFi=False, GSM=true) as the important ones. The tool then automatically adds the smallest possible set of tests that cover these holes (and as many of the unselected ones as possible).

Alternatively, if the resources for running more tests are still limited, the user may limit the tool in the number of tests it is allowed to choose. In this case, the user may either instruct the tool to cover as many of the selected holes as possible by adding tests within the specified limit, or otherwise define priorities for covering the holes, and instruct the tool to try and cover more important ones first.

C. Modifying Tests

Another common operation is the modification of certain values in a test. One scenario in which such modifications

may be necessary is when complex restrictions exist between parameters. Some users tend to omit such restrictions from the input model, and manually "fix" any invalid test in the result. Another common scenario is when certain tests are easier to execute, or otherwise preferable over others.

In Figure 4, the GUI for modifying a test is shown. In this example, the value for CDMA support in the sixth test is modified from true to false. Similarly to the add test dialog in Figure 2, the right pane previews the effect of this modification. Specifically, two holes will get covered (CDMA=false, numCameras=0 and GPS=true, CDMA=false), but a new one will be introduced (CDMA=true, touchscreen=false).

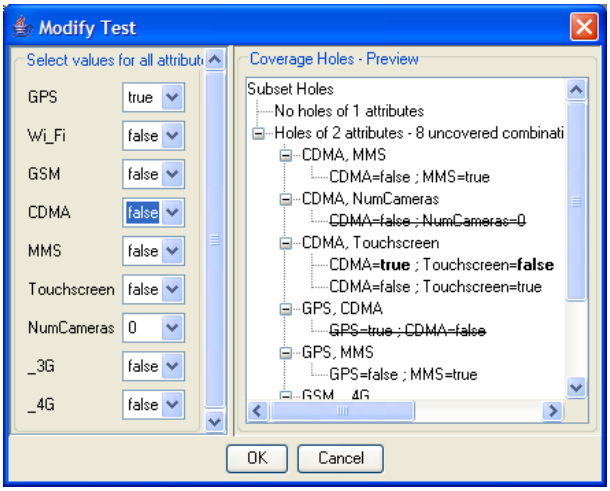

Figure 4. The dialog for modifying a test, showing the updated values in the test, and a preview of its effect on the holes analysis.

D. Changing Requirements

Finally, a test architect may wish to examine a certain test plan against different sets of interaction coverage requirements. These may be changed in the bottom part of the window, and the holes analysis will immediately be updated accordingly, to reflect holes in the current test plan, when considered against the updated requirements.

IV. Future Work

In this paper, we discuss a novel feature in IBM's test planning tool, which allows interactive modification of a combinatorial test plan, while viewing its coverage gaps. Preliminary experience with our users already shows great value, though further empirical study is needed to systematically analyze its usage and points for improvement.

Specifically, some users expressed interest in the ability to document the modifications that were performed, along with textual description thereof. In addition, we plan to further investigate and optimize the algorithmic issues involved in adding tests to cover specific holes. These require non-trivial algorithmic support and are out of scope for this paper.

References

[1] 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 *ICSE'99*, pages 285–294, 1999.

[2] M. Grindal, J. Offutt, and S. F. Andler. Combination Testing Strategies: A Survey. *Software Testing, Verification, and Reliability*, 15:167–199, 2005.

[3] https://www.research.ibm.com/haifa/video/IntRefDemo.htm.

[4] 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.

[5] Pairwise testing website. http://www.pairwise.org.

[6] I. Segall, R. Tzoref-Brill, and E. Farchi. Using Binary Decision Diagrams for Combinatorial Test Design. In *ISSTA'11*, pages 254–264, 2011.

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