

# Using Combinatorial Test Models to Improve Software Testing Efficiency

## ***Abstract***

Software is virtually omnipresent in today's world. It has significant impact to the quality of life since the performance of key institutions of modern society depends on their ability to manage the software lifecycle.

Despite advancements in supporting technologies, the outcome of software product development is difficult to control, comes with an expensive price tag and, although a significant proportion of the cost of software projects is allocated for testing, failures still occur with the finished deliverables. An often cited report draws the alarming conclusion that inadequate software testing incurs a cost of \$59.5 billion annually [NS02]. An increasing number of industry observers denounce this as a software crisis, noting that progress is no longer viable using conventional methods; new approaches are required to overcome the current stalemate.

A contrasting analysis of some old, new and "revolutionary" directions in software development seems to suggest that, beyond differences in the philosophical fundament, they commend one way or another, an iterative, incremental development in a controlled environment. This practical, common sense strategy is simply a reflection of the way humans learn and solve difficult problems. And software testing is no exception of this.

This paper describes a practical solution to improve software testing efficiency through a model-driven approach. It can be used standalone or applied in addition to other techniques by reusing test cases determined with other methods addressing most used or riskiest scenarios. It creates concise artifacts that allow for easy auditing and effective review of a large number of test cases: instead of reviewing cases one by one, the same effect is achieved by reviewing the rules that generated them.

The proposed solution was designed as a thin wrapper of current combinatorial, pair-wise techniques which have been long used to provide a systematic, statistical way of creating test case inputs for scenarios where exhaustive testing is virtually impossible to conduct. It naturally provides support for implementation of decoupled models, an efficient method of eliminating ineffective pairing of independent test case inputs. The solution bundles a tool that is easy to use to author models and to transform them into test cases inputs. With this tool, models can be developed incrementally, morphed or split, with the impact of each step immediately quantified and available for analysis. As such, models are truly used to direct the course of understanding, design and implementation of software testing. The net result is determined by the effectiveness of combinatorial techniques applied to software testing, which research seems to indicate is 96% as effective as exhaustive testing, while typically using 95% less test cases.

## ***Copyrights and Trademarks***

© 2004-2008 QTAssistant.com. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of QTAssistant.com. The information contained herein may be changed without prior notice.

Some software products marketed by QTAssistant.com and its distributors contain proprietary software components of other software vendors.

Microsoft, Microsoft Windows and Microsoft Internet Explorer are trademarks, or registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Solaris is a trademark or registered trademark of Sun Microsystems, Inc. in the United States, other countries or both.

Macintosh is registered trademark and Safari is a trademark of Apple Computer, Inc. in the United States other countries, or both.

Linux is a trademark of Linus Torvalds.

IBM and Rational are trademarks of International Business Machines Corporation in the United States, other countries, or both.

OMG and UML are trademarks of the Object Management Group.

Other company, product, or service names may be trademarks or service marks of others.

# Combinatorial Test Models

## Overview

A Combinatorial Test Model (CTM) defines a view of the system under test which can be used for automatic generation of Data Pools and Test Case kernels. A CTM can be created from scratch or using Entity-Relationship models, W3C XML Schema or UML. The CTM foundation is the practical application of mathematical covering arrays theory to interaction software testing.

## Background

Many domains of activity are faced with problems where the ideal solution would consist of exhaustive exploration of all possible combinations of the problem inputs. Other than trivial problems, this approach is usually not feasible. For software testing, some of the solutions currently used to address this problem are based on combinatorial techniques.

Combinatorial techniques are rooted in orthogonal arrays. The mathematical background of orthogonal arrays is beyond the scope of this paper. However, selected aspects will be presented in this section just to facilitate the understanding of this paper. In many cases, established mathematical notations have been abandoned in favor of plain language and examples have been called in support of theory.

An orthogonal array is a two-dimensional array, with rows and columns like a table in a database. Each column represents a parameter or *factor*. Each parameter has the same number of possible values, referred to as the *level*. Each row represents a possible combination of parameters values. The number of factors for which all possible combinations are found an equal number of times within the array is called *strength*. Table 1 shows an orthogonal array for five factors, each factor with two possible values and all possible two-way combinations covered exactly twice.

A0	B0	C0	D0	E0
A1	B0	C0	D1	E1
A0	B1	C0	D1	E0
A0	B0	C1	D0	E1
A1	B1	C0	D0	E1
A1	B0	C1	D1	E0
A0	B1	C1	D1	E1
A1	B1	C1	D0	E0

**Table 1 Orthogonal array with 5 factors, 2 levels and strength 2**

For many practical applications, orthogonal arrays have been deemed too restrictive. Covering arrays differ from orthogonal arrays in that the strength is now determined if all possible combinations are found at least once. Table 2 shows a covering array that has the same specification as the orthogonal array shown in Table 1. However, the number of rows in the covering array is less than the one in the orthogonal array because pairs now have to show at least once.

A0	B1	C1	D0	E0
A1	B0	C0	D1	E1
A0	B0	C0	D1	E0
A1	B1	C1	D1	E1
A1	B1	C0	D0	E1

A1	B0	C1	D0	E0
A0	B0	C0	D1	E1

**Table 2 Covering array with 5 factors, 2 levels and strength 2**

Parameters in general do not have exactly the same number of levels. Mixed level covering arrays (sometimes simply referred to as mixed arrays) differ from covering arrays in that each parameter may have any number of values. Table 3 shows an array with 5 factors, with 2, 3, 2, 4, and 3 levels respectively, and strength 2.

A0	B2	C1	D0	E1
A1	B1	C0	D2	E2
A1	B0	C1	D3	E0
A0	B0	C0	D1	E1
A0	B1	C0	D0	E0
A1	B2	C1	D1	E2
A0	B2	C0	D2	E0
A0	B0	C0	D3	E2
A1	B1	C1	D2	E1
A1	B0	C1	D0	E2
A1	B1	C0	D3	E1
A0	B1	C1	D1	E0
A0	B0	C0	D2	E1
A0	B2	C0	D3	E0

**Table 3 Mixed array with 5 factors and strength 2**

The total number of possible combinations for a given set of parameters is calculated by multiplying together all the levels. The minimum number of combinations required for strength  $n$  ( $n \geq 2$ ) is calculated by multiplying together the first  $n$  levels sorted in descending order. For the specification in Table 3:

- All possible combinations are 144 in total.
- The number of covering combinations for different strengths is shown in Table 4. Increased strength yields more combinations with better coverage.

Strength	Total combinations
2	14
3	40
4	81
5	144

**Table 4 Total number of combinations as a function of strength**

- The minimum number of covering combinations for different strengths is shown in Table 5.

Strength	Minimum combinations
2	12
3	36
4	72

**Table 5 Minimum number of combinations as a function of strength**

When applied to software testing, these are some of the most frequently asked questions:

- What is the appropriate strength to use with the array? “Appropriate” could be defined as the point of diminishing returns at which using an  $n$ -strength array is nearly as effective as an  $n+1$ -strength array. [DE02] suggests that “[...] more than 95% of errors in the software studied would be detected by test cases that cover all 4-way combinations of values”, while concluding that appropriate strengths could be between 3 and 6, according to dependability requirements.

- What goes as parameters into an array? Since the whole technique provides for interaction testing, parameters that are meshed together through some dependency should also share an array. Test cases involving one or more independent parameters would prove useless since the effect of other parameters on the outcome is not influenced by pairing with these independent parameters.
- What is the best way to handle high risk areas of test case inputs sharing the same array? It depends on many factors, like the number of parameters involved, levels, and the strengths considered. One approach would be to extend the high risk strength to the whole array. The other could be to have high risk areas assigned to another array with the appropriate, higher strength. The output of this array could then be added as a parameter back in the original array. As an example, A, C and D from Table 3 require all possible combinations (strength 3). As shown in Table 4, strength 3 applied to the whole array yields 40 combinations. Separating A, C and D in another array for all possible combinations, yields 16 combinations. Having a 16-levels parameter combined with B and E with a strength of 2, yields 48 combinations (somewhat expected, since this is the minimum number for this scenario). For all reasons, in this case would be better to go with the first solution that provides better overall coverage with fewer test cases. Let's assume another scenario: add two more 3-level factors, E and F to Table 3; have A, B, C and D now requiring all possible combinations. Strength 4 applied to the modified array yields 126 combinations. Using two arrays approach, yields 96 combinations (using strength 4 and 2). If the best solution is the one with fewer combinations, this scenario benefits from the second approach. It is recommended to experiment since there are two many variables to consider. In general, splitting arrays is recommended when the original array has strength 2, and high risk areas require strength greater than 3.

For the rest of this document, “covering array” is used to also refer to “mixed array”, unless otherwise noted.

## ***Basic concepts***

The following list summarizes concepts CTM makes reference to and defined here [TP05]<sup>1</sup>.

Test Context	A collection of test cases together with a test configuration on the basis of which the test cases are executed.
SUT	The system under test (SUT) is the entity being tested. The SUT is exercised via its accessible interaction points by the testing probes. No further information can be obtained from the SUT as it is a black-box.
Test Objective	A test objective describes what should be tested and it is associated with a test case.
Test Case	A test case is a specification of one case to test the system including what to test with, which input, result, and under which conditions. It is a complete technical specification of how the SUT should be tested for a given test objective. A test case is defined in terms of sequences, alternatives, loops, and defaults of stimuli to and observations from the SUT. It implements a test objective.
Stimulus	Test data sent to the SUT in order to control it and to make assessments about the SUT when receiving the SUT reactions to these stimuli.
Data Partition	A logical value for a part used in a stimulus or in an observation. It typically defines an equivalence class for a set of values (e.g., valid user names, etc.).
Data Pool	A data pool is a collection of data partitions or explicit values that are used during the evaluation of test cases. In doing so, a data pool offers a means for providing values or data partitions for repeated tests.

---

<sup>1</sup> Some definitions have been altered to avoid references to concepts not used in here. For in-depth studying, the understanding of the original text is recommended.

**Wildcard** Wildcards are special symbols to represent values or ranges of values. Wildcards allow the user to explicitly specify whether the value is present or not, and/or whether it is of any value. Wildcards are used instead of symbols within instance specifications. Three wildcards exist: a wildcard for any value, a wildcard for any value or no value at all (i.e. an omitted value), and a wildcard for an omitted value.

The following list summarizes additional concepts.

Covering Array	<p>A covering array is basically a two-dimensional array. It is described by the following fundamental properties:</p> <ul style="list-style-type: none"> <li>○ <math>N</math> – the number of rows</li> <li>○ <math>k</math> – the number of columns</li> <li>○ <math>(v_i)</math> – a set of numbers; <math>v_i</math> is the <math>i</math>-th parameter level, representing the number of distinct values recorded in the <math>i</math>-th column.</li> <li>○ <math>t</math> – the interaction strength (simply referred to as strength)</li> </ul> <p><math>N_{\max}</math> is the maximum value for <math>N</math> and can be calculated by multiplying together all <math>v_i</math></p>
Covering Array Definition	A covering array definition is a named element describing a covering array, including its fundamental properties, a collection of constraints and references to other covering arrays. It is a complete technical specification of how to build a covering array.
Parameter	A parameter is a uniquely identifiable part used in a stimulus. It is a column of a covering array. Using types defined by the UML Model, a parameter can be associated with an enumeration.
Placeholder	A placeholder is a subset of the complete set of parts used in a stimulus. It is associated with a distinct covering array definition. The placeholder's domain is a covering array. An example of practical applicability is to vary the interaction strength for the subset.
Value	A value represents a unique state of a part used in a stimulus. It is associated with a parameter. Using types as defined by the UML Model, a value is the literal describing possible values of the enumeration.
Value Placeholder	A value placeholder is uniquely associated with a covering array.
Domain	A domain is a collection of data partitions or explicit values that are used during the evaluation of test cases by a parameter or placeholder.
Tuple	A tuple is a row, or a subset of a row, of a covering array. The notation $n$ -tuple denotes a tuple with $n$ columns.
Strength	The strength of the coverage array is an integer value describing the maximum number of parameters ( $n$ ) with the property that all possible ordered $n$ -tuples occur at least once. It has a minimum value of 2 and a maximum value of $k$ . When the strength value equals $k$ , $N$ is equal to $N_{\max}$ . For practical reasons, it is not recommended to use a value greater than 6.
Inclusion	Inclusion is a particular type of constraint applicable to a covering array. It describes completely one or more rows that must be present in the covering array. It is associated with a covering array definition.
Exclusion	Exclusion is a particular type of constraint applicable to a covering array. It describes completely one or more rows that must not be present in the covering array. It is associated with a covering array definition.

Target Context      A target context is a literal value intended to be used as a variable in a filter. It can be associated with a covering array, parameter, placeholder, value, inclusion and exclusion. Possible uses are:

- Reuse same model in various test contexts
- Create test cases targeting different equivalence classes of test objectives (“positive”, “negative”, etc.)

## Concept examples

### Problem

An internet based system, the SUT, has to support a variety of browsers, running off different platforms and using different network connections to their Internet Service Providers. The requirement is to provide a statistically comprehensive set of test cases, without going through all possible combinations of inputs.

**Note:** To keep the illustrations brief, yet relevant, we’ve narrowed the selection of possible values to just a few. The downside of this is in the relatively low percentage of “savings” (around 50%) displayed in the reduction of test cases, as compared to the total number of possible combinations. By choosing a wider selection, closer to what is the current [state](#) of computing industry vis-à-vis internet browsers and platforms, the recorded savings could exceed 95% test cases for strength 2.

### Solution

The parameters and values used by the covering array definition are listed in the table below.

<i>Parameters</i>			
<i>Values</i>	<b>Web Browser</b>	<b>Platform</b>	<b>Network Connection</b>
	Internet Explorer	Windows	LAN
	Safari	Macintosh	Dialup
	Mozilla Firefox	Linux	
		Solaris	

**Table 6** Covering array definition (sample)

The maximum number of all possible combinations is 24 (3 x 4 x 2). The minimum number of combinations for strength 2 is 12 (4 x 3).

Different construction algorithms used for covering arrays may generate a different numbers of rows. The following is a possible covering array (with no constraints) for the provided definition. It is represented as a table with a heading row and an additional column **ID** attached for identification purposes.

<b>ID</b>	<b>Web Browser</b>	<b>Platform</b>	<b>Network Connection</b>
1	Internet Explorer	Solaris	LAN
2	Safari	Macintosh	Dialup
3	Mozilla Firefox	Windows	LAN
4	Safari	Linux	LAN
5	Internet Explorer	Linux	Dialup
6	Mozilla Firefox	Macintosh	LAN
7	Mozilla Firefox	Linux	Dialup
8	Internet Explorer	Windows	Dialup

ID	Web Browser	Platform	Network Connection
9	Safari	Solaris	Dialup
10	Internet Explorer	Macintosh	Dialup
11	Safari	Windows	LAN
12	Mozilla Firefox	Solaris	Dialup

**Table 7 Covering array (sample)**

However, in real life, “Internet Explorer” is not running on “Linux” or “Solaris”. Safari also is a Mac-running browser only. By placing constraints on the definition, certain test cases can be avoided.

“Internet Explorer” restrictions may be represented<sup>2</sup> as an “Exclusion” {“Web Browser”: [“Internet Explorer”], “Platform”: [“Linux”, “Solaris”]} meaning that test cases using inputs containing any of (“Internet Explorer”, “Linux”) or (“Internet Explorer”, “Solaris”), are prohibited.

Just to show how a wider coverage can be achieved with one exclusion, the expression {“Web Browser”: [“Internet Explorer”, “Safari”], “Platform”: [“Linux”, “Solaris”]} dismisses inputs containing any of (“Internet Explorer”, “Linux”), (“Internet Explorer”, “Solaris”), (“Safari”, “Linux”) or (“Safari”, “Solaris”).

The constrained covering array is shown in Table 8.

ID	Web Browser	Platform	Network Connection
1	Internet Explorer	Macintosh	LAN
2	Safari	Windows	Dialup
3	Mozilla Firefox	Solaris	Dialup
4	Mozilla Firefox	Linux	LAN
5	Internet Explorer	Windows	Dialup
6	Safari	Macintosh	LAN
7	Mozilla Firefox	Windows	LAN
8	Mozilla Firefox	Macintosh	Dialup
9	Mozilla Firefox	Linux	Dialup
10	Mozilla Firefox	Solaris	LAN

**Table 8 Constrained covering array (sample)**

Table 8 still contains one invalid test case (“Safari”, “Windows”), which can be addressed by adding the exclusion {“Web Browser”: [“Safari”], “Platform”: [“Windows”]}. The final and correct set of test cases is shown in Table 9.

ID	Web Browser	Platform	Network Connection
1	Internet Explorer	Macintosh	LAN
2	Mozilla Firefox	Windows	LAN
3	Safari	Macintosh	Dialup
4	Mozilla Firefox	Linux	Dialup
5	Mozilla Firefox	Solaris	Dialup
6	Safari	Macintosh	LAN
7	Internet Explorer	Windows	Dialup
8	Mozilla Firefox	Linux	LAN
9	Mozilla Firefox	Macintosh	LAN
10	Mozilla Firefox	Solaris	LAN

**Table 9 Constrained and complete with real-life exclusions covering array (sample)**

<sup>2</sup> A future revision of this paper will seek alignment with UML 2 OCL available specifications.



An “Inclusion” can be represented as {“Web Browser”: “Internet Explorer”, “Platform”: “Macintosh”, “Network Connection”: “Dialup”}.

## **Conclusion**

The solution can be gradually determined through an iterative and simple procedure.

1. Determine parameters and input values.
2. Select the covering array strength.
3. Analyze the resulting covering array and amend its definition with inclusions and/or exclusions as needed.
4. Reiterate from any step above.

This routine is applicable for non-trivial problems in equal measure.

## ***Relationships between concepts***

This section summarizes some important relationships between the defined concepts.

- The number of rows in the covering array is equal to the number of test cases which should be executed.
- A row in a covering array provides the inputs of a test case.
- A test objective for each test case is inferable through analysis of each row in a covering array.

## ***Combinatorial Test Model Diagram***

A CTM Diagram revolves around five simple graphical elements: covering array definition, parameter, value, placeholder and link.

## **Covering Array Definition**

This element represents the Covering Array Definition concept. An example with a structure that matches Table 6 content is depicted by Figure 1.



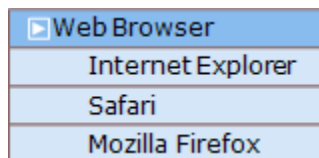
**Figure 1 Graphical notation for the Covering Array Definition concept**

## Operation List

- Delete
- Add Parameter
- Remove Parameter
- Add Placeholder
- Remove Placeholder
- Modify Strength
- Add Target Context
- Remove Target Context
- Change Label
- Add Inclusion
- Remove Inclusion
- Add Exclusion
- Remove Exclusion
- Link with a Placeholder (one only)
- Drop Link

## Parameter

This element represents the Parameter concept. An example with a structure that matches the Web Browser parameter in Table 6 is depicted by Figure 2.



**Figure 2 Graphical notation for the Parameter concept.**

## Operation List

- Delete
- Add Value
- Remove Value
- Add Target Context
- Remove Target Context
- Change Label

## Value

This element represents the Value concept. An example with a structure that matches the “Internet Explorer” value, of the “Web Browser” parameter, listed in Table 6 is depicted by Figure 3.

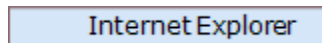


Figure 3 Graphical notation for the Value concept

## Operation List

- Delete
- Add Target Context
- Remove Target Context
- Change Label

## Placeholder

This element represents the Placeholder concept in its unlinked state. It is depicted by Figure 4.

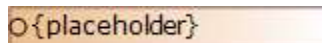


Figure 4 Graphical notation for the Placeholder concept

## Operation List

- Delete
- Accept Link from another Covering Array (one only).
- Drop Link

## Link

This element realizes a relationship between two covering arrays. A link between two covering arrays is shown in Figure 5. The numeric value label displayed with the link represents the  $N$  of the source covering array.

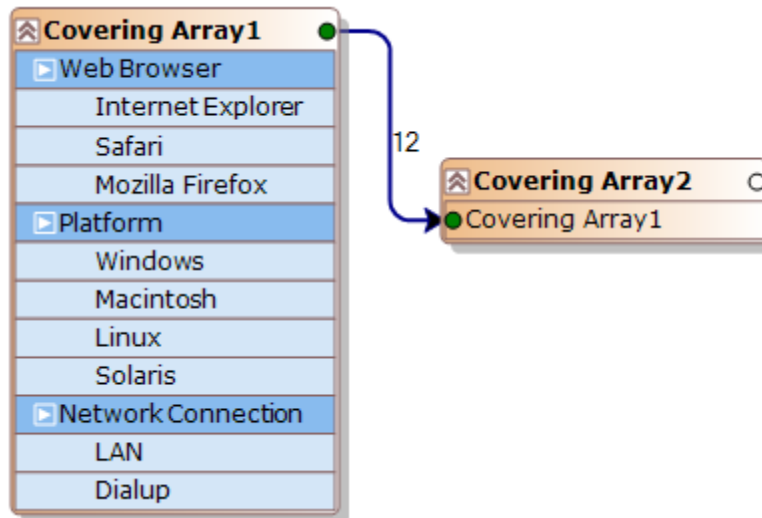


Figure 5 A link between two covering arrays

## Operation List

- Delete

# An Example Case Study: Web-based Mortgage Calculator

To illustrate the proposed approach, a sample application has been designed and built, inspired by a real life example. The requirements were selected such that the complexity of the application to require a non-trivial combinatorial test model, yet uncluttered, easy to follow and relevant to a wide range of other applications. Integration with IBM Rational Functional Tester is also provided, to address integration with automation tools.

## Mortgage Calculator High Level Requirements

The major use case is “Fill in the Mortgage Scenario”. The main screen parts supporting this use case are shown in Figure 6 and Figure 7.

Figure 6 Mortgage Calculator GUI overview

Figure 7 Other Payment Options Input Overview

## Requirements summary

- The Mortgage Calculator must support user inputs as listed in Table 10.

Short Description	Notes
Mortgage amount	User entered. Defaults to \$100,000.00
Product Category	User entered. Defaults to Fixed.
Term	User entered. Defaults to 60 months.
Promotional Rate	User entered. Defaults to 0.
Promotional Period	User entered. Defaults to 0.
Interest Rate	User entered. Defaults to 5.
Proposed Amortization	User entered. Defaults to 25 years.
Payment Frequency	User entered. Defaults to Monthly.
Promotional Payment	The user may temporarily override the system calculated value.
Payment	The user may temporarily override the system calculated value.
Funding Date	User entered or determined from Interest Adjustment Date.
Interest Adjustment Date	User entered or determined from First Payment Date or from Funding Date.
First Payment Date	User entered or determined from Interest Adjustment Date.
Payment Acceleration	User entered. Defaults to None.
Payment Round Up	User entered. Defaults to None.
Payment Increase Percentage Option	User entered. Defaults to 0.
Payment Increase Amount Option	User entered. Defaults to 0.
Payment Increase Timing Option	User entered. Defaults to Once.

**Table 10 List of user inputs**

- The Mortgage Calculator must calculate and display values as listed in Table 11.

Short Description	Notes
Number of payments during the Term	A function of payment frequency and term duration.
Number of payments during the Proposed Amortization	A function of payment frequency and amortization duration.
Promotional Payment	Represents a minimum value that matches the proposed amortization. A function of mortgage amount, payment frequency, promotional rate, amortization duration, product category.
Payment	Represents a minimum value that matches the proposed amortization. A function of mortgage amount, payment frequency, rate, amortization duration, product category.
Promotional Payment w/ Options applied	A function of options and promotional payment.
Payment w/ Options applied	A function of options and payment.
Funding Date	As a function of Interest Adjustment Date
Interest Adjustment Date	As a function of Funding Date or First Payment Date
First Payment Date	As a function of Interest Adjustment Date
Promo Expiry Date	As a function of Funding Date and Promotional Period
Maturity Date	As a function of Interest Adjustment Date and Term
Balance At Maturity	A function of term, rate, payment(s).
Remainder Amortization (duration)	For convenience, the duration is represented as whole years and months.

Short Description	Notes
Remainder Amortization (payments)	Remainder amortization duration represented as payments. A function of remainder amortization duration and payment frequency.
Interest To Maturity (end of Term)	
Interest To Amortized Mortgage	Assumes current conditions until the mortgage is fully amortized (remaining balance zero).
Revised Amortization (duration)	For convenience, the duration is represented as whole years and months.
Revised Amortization (payments)	Revised amortization represented as payments. A function of revised amortization duration and payment frequency.

**Table 11 List of calculated user fields**

- A product category selection drives the availability of input fields according to Table 12.

High level category features	Promotional Rate	Promotional Period	Promotional Payment
Fixed Rate Single Rate/Single Payment	(N/A)	(N/A)	(N/A)
Variable Rate Single Rate/Single Payment	(N/A)	(N/A)	(N/A)
Fixed Rate Promotional Rate/Payment Ongoing Rate/Payment	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Variable Rate Promotional Rate Single Payment	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	(N/A)

**Table 12 Input fields availability based on Product Category selection**

- The system must provide automatic date adjustments between “Funding/Renewal Date”, “IAD/Last Payment Date” and “First Payment Date”. (For the purposes of this paper, it is assumed that these fields are functioning as a pseudo-radio buttons group.)
- The Other Payment Options panel will exhibit the following behavior:
  - Accelerate and Round Up must function independently of each other and the rest of panel’s fields; the user may opt-out of both.
  - Percentage Payment Increase Option is mutually exclusive with Amount Payment Increase Option. The user may opt-out of both.
  - The timing (“Once” or at “Anniversaries”) of a Payment Increase Option is applicable only if a non-zero valued Payment Increase Option has been specified.
- An adjustment to any input field will trigger recalculations of all other fields.

## ***Combinatorial Test Model-based solution design***

### **Define the Covering Array Definition for the date input fields**

Mutually exclusive parameters can be modeled as values of one surrogate parameter. Given the mechanics of the combinatorial test models, only one of parameter’s values is used in a test at any given time, hence,

the mutually exclusive condition is fulfilled by definition. One possible implementation is shown in Figure 8.

▶Funding/IAD/First Pymt Dates
funding::2006-02-28
funding::2006-02-24
iad::2006-03-03
firstpymt::2006-03-15

**Figure 8 The "Funding/IAD/First Payment" implemented as Parameter**

This approach is easy to model and could be considered structured and machine parse-able when using rigorous notational conventions. However, when considering the whole picture (for manual or automated testing), the savings in the modeling area is probably quickly offset by the overhead required to consume the result. The generated set of test cases now contains “overloaded” columns. An overloaded column is one that changes its binding to another input field as a function of the value in the cell.

A more robust model could be developed by using the alternative to the above, i.e. constructing a Covering Array Definition instead. The parameters will match the input fields (see Figure 9).

⤴ Funding/IAD/First Py...O
▶Funding
2006-02-28
2006-02-24
▶IAD
2006-03-03
▶FirstPymt
2006-03-15

**Figure 9 The "Funding/IAD/First Payment" implemented as Covering Array Definition (1<sup>st</sup> iteration)**

The problem with this first iteration stays with the inability of this Covering Array Definition to deliver on the mutually exclusive condition. To assist with this, a new value (N/A) will be added to the value set of each parameter (see Figure 10).

⤴ Funding/IAD/First Py...●
▶Funding
(N/A)
2006-02-28
2006-02-24
▶IAD
(N/A)
2006-03-03
▶FirstPymt
(N/A)
2006-03-15

**Figure 10 The "Funding/IAD/First Payment" Covering Array Definition**



A quick look at the proposed test cases (Figure 11) will show that we have more than the expected four combinations, with some invalid. The solution to this is to place appropriate constraints and monitor the outcome of each. One constraint is to enforce mutually exclusiveness between “Funding” and “IAD” parameters (see Figure 12 for rule implementation and results). Following the same principle, two new constraints are added for “Funding” and “First Payment” and “IAD” and “First Payment” (Figure 13 and Figure 14, respectively). Figure 14 shows that the expected result has now been achieved.

Test Case	Funding	IAD	FirstPymt
1	(N/A)	2006-03-03	(N/A)
2	2006-02-28	(N/A)	2006-03-15
3	2006-02-24	(N/A)	(N/A)
4	2006-02-28	2006-03-03	2006-03-15
5	(N/A)	(N/A)	2006-03-15
6	2006-02-24	2006-03-03	2006-03-15
7	2006-02-28	2006-03-03	(N/A)

Exclusions

Inclusions

Description

Funding

IAD

FirstPymt

**Figure 11 Unconstrained test cases for the "Funding/IAD/First Payment" Covering Array Definition**

Test Case	Funding	IAD	FirstPymt
1	(N/A)	2006-03-03	(N/A)
2	2006-02-28	(N/A)	2006-03-15
3	2006-02-24	(N/A)	(N/A)
4	(N/A)	(N/A)	2006-03-15
5	2006-02-24	(N/A)	2006-03-15
6	2006-02-28	(N/A)	(N/A)
7	(N/A)	2006-03-03	2006-03-15

Exclusions

Inclusions

Rule

Description

Specifying a Funding Date deems all others N/A

Funding

2006-02-28,2006-02-24

IAD

2006-03-03

FirstPymt

**Figure 12 Specifying constraints to make Funding and IAD mutually exclusive.**

Test Case	Funding	IAD	FirstPymt
1	(N/A)	2006-03-03	(N/A)
2	2006-02-28	(N/A)	(N/A)
3	2006-02-24	(N/A)	(N/A)
4	(N/A)	(N/A)	2006-03-15
5	(N/A)	2006-03-03	2006-03-15

Exclusions			
		Rule	Rule
Inclusions	Description	Specifying a Funding Date deems all others N/A	Specifying a Funding Date deems all others N/A
	Funding	2006-02-28,2006-02-24	2006-02-28,2006-02-24
	IAD	2006-03-03	
	FirstPymt		2006-03-15

**Figure 13 Specifying constraints to make Funding and First Payment mutually exclusive.**

Test Case	Funding	IAD	FirstPymt
1	(N/A)	2006-03-03	(N/A)
2	2006-02-28	(N/A)	(N/A)
3	2006-02-24	(N/A)	(N/A)
4	(N/A)	(N/A)	2006-03-15

Exclusions				
		Rule	Rule	Rule
Inclusions	Description	Specifying a Funding Date deems all others N/A	Specifying a Funding Date deems all others N/A	Specifying IAD takes out First Pymt date
	Funding	2006-02-28,2006-02-24	2006-02-28,2006-02-24	
	IAD	2006-03-03		2006-03-03
	FirstPymt		2006-03-15	2006-03-15

**Figure 14 Specifying constraints to make IAD and First Payment mutually exclusive.**

## Define the Covering Array Definition for other payment options input fields

Given the requirements for the Payment Options Panel, two Covering Array Definitions had been created.

## Define the Covering Array Definition for Increases

This Covering Array Definition is built on the same principle as the “Funding/IAD/First Payment” since mutually exclusiveness between Percentage and Amount is required. However, the element is extended to include the Timing input field and the applicability conditions around it (see Figure 15).

Increases	
▶ Percentage	
	(N/A)
	3.5%
▶ Amount	
	(N/A)
	\$50.00
▶ When	
	(N/A)
	OnceOnly
	Anniversaries

**Figure 15 The “Increases” Covering Array Definition**

Without any constraints, the resulting test cases are shown in Figure 16.

Test Case	Percentage	Amount	When
1	(N/A)	\$50.00	(N/A)
2	3.5%	(N/A)	Anniversaries
3	(N/A)	(N/A)	OnceOnly
4	3.5%	\$50.00	OnceOnly
5	3.5%	(N/A)	(N/A)
6	(N/A)	\$50.00	Anniversaries

Exclusions	

Inclusions	
▶	Description
	Percentage
	Amount
	When

**Figure 16 Test cases generated by an unconstrained “Increases” Covering Array Definition**

The rules required to achieve the expected results are summarized bellow:

- Figure 17 – The Timing input field is mandatory for non-zero Percentage
- Figure 18 – The Timing input field is mandatory for non-zero Amount
- Figure 19 – The Timing input field is not applicable when Percentage and Amount is not specified.
- Figure 20 – Percentage and Amount fields are mutually exclusive.

The result depicted in Figure 20 is now a valid set of test cases.

Test Case	Percentage	Amount	When
1	(N/A)	\$50.00	(N/A)
2	3.5%	(N/A)	Anniversaries
3	(N/A)	(N/A)	OnceOnly
4	3.5%	\$50.00	OnceOnly
5	(N/A)	\$50.00	Anniversaries
6	(N/A)	(N/A)	(N/A)

Exclusions

	Rule
►  Description	An increase must be timed when it happens.
Percentage	3.5%
Amount	
When	(N/A)

Inclusions

Figure 17

Test Case	Percentage	Amount	When
1	(N/A)	\$50.00	OnceOnly
2	3.5%	(N/A)	Anniversaries
3	(N/A)	(N/A)	(N/A)
4	3.5%	\$50.00	Anniversaries
5	3.5%	(N/A)	OnceOnly
6	(N/A)	(N/A)	Anniversaries

Exclusions

Inclusions

		Rule	Rule
▶	Description	An increase must be timed when it happens.	An increase must be timed when it happens.
	Percentage	3.5%	
	Amount		\$50.00
	When	(N/A)	(N/A)

Figure 18

Test Case	Percentage	Amount	When
1	(N/A)	\$50.00	Anniversaries
2	3.5%	(N/A)	OnceOnly
3	(N/A)	(N/A)	(N/A)
4	3.5%	\$50.00	Anniversaries
5	(N/A)	\$50.00	OnceOnly
6	3.5%	(N/A)	Anniversaries

Exclusions

Inclusions



		Rule	Rule	Rule
	 Description	An increase must be timed when it happens.	An increase must be timed when it happens.	An increase must be timed when it happens.
	Percentage	3.5%		(N/A)
	Amount		\$50.00	(N/A)
	When	(N/A)	(N/A)	OnceOnly,Anniversaries

Figure 19

Test Case	Percentage	Amount	When
1	(N/A)	\$50.00	Anniversaries
2	3.5%	(N/A)	OnceOnly
3	(N/A)	(N/A)	(N/A)
4	(N/A)	\$50.00	OnceOnly
5	3.5%	(N/A)	Anniversaries

Exclusions

Inclusions

		Rule	Rule	Rule	Rule
	 Description	An increase must be timed when it happens.	An increase must be timed when it happens.	An increase must be timed when it happens.	Percentage and Amount are mutually exclusive.
	Percentage	3.5%		(N/A)	3.5%
	Amount		\$50.00	(N/A)	\$50.00
▶	When	(N/A)	(N/A)	OnceOnly,Anniversarie	

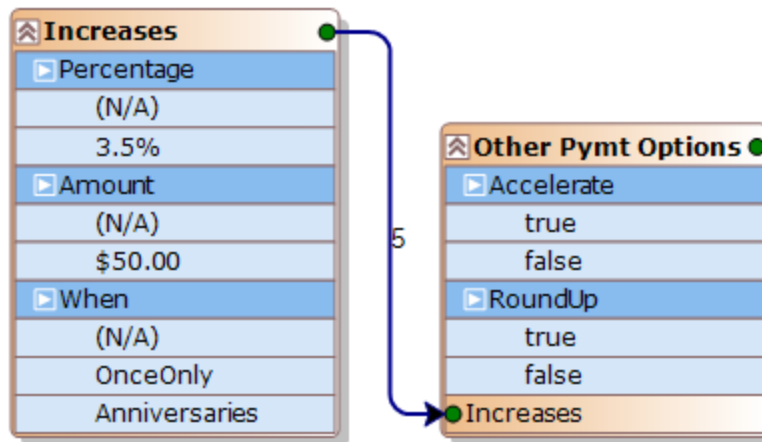
☐ (N/A)  
☒ OnceOnly  
☒ Anniversaries

Close

Figure 20

## Define the “Other Payment Options” Covering Array Definition

This new element is using the output from “Increases” Covering Array Definition with the rest of the input fields from the Other Payment Options panel, namely “Accelerate” and “Round Up” (see Figure 21).



**Figure 21 The “Other Payment Options” Covering Array Definition**

The resulting test cases for “Other Payment Options” are shown in Figure 22.

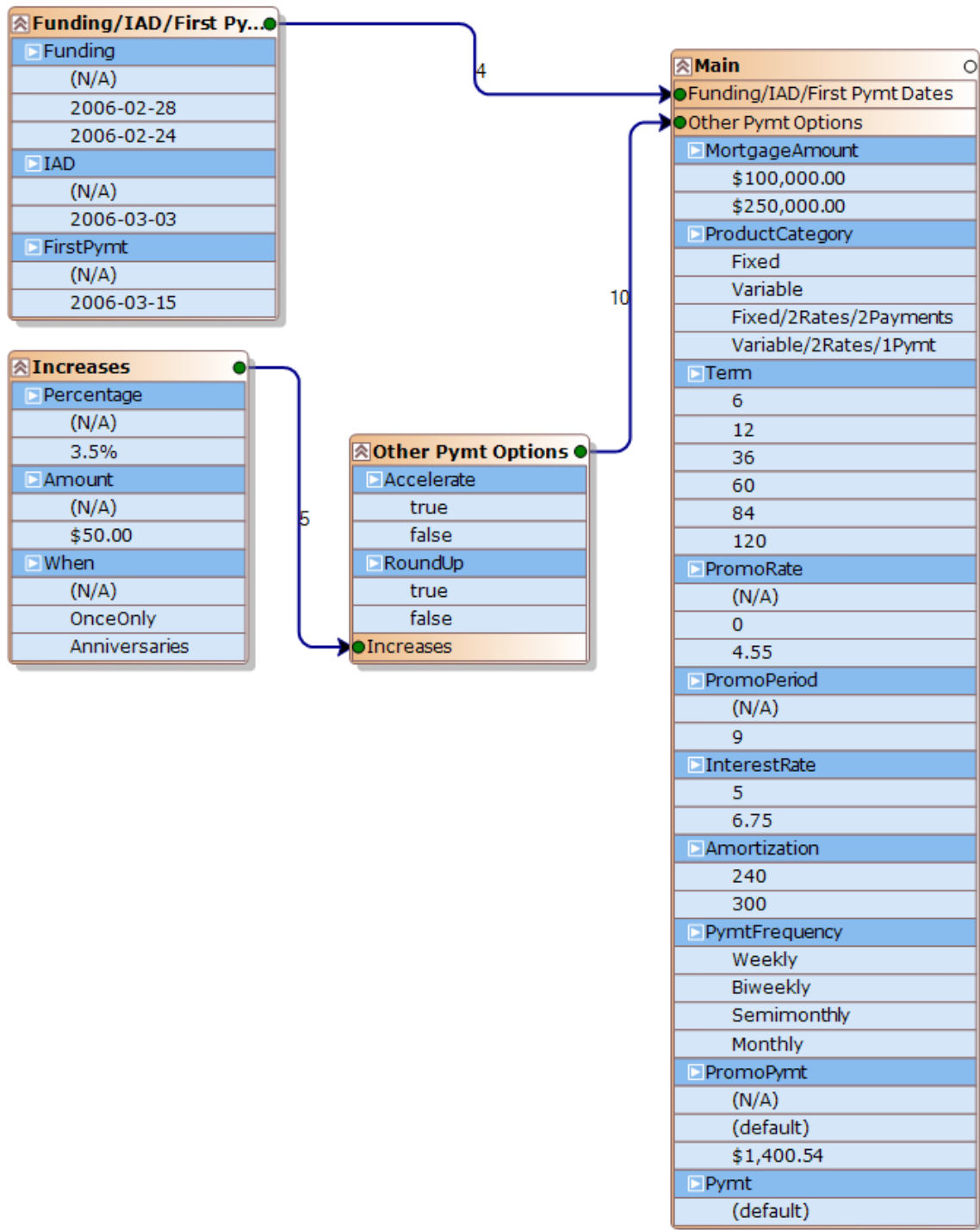
Test Case	Accelerate	RoundUp	Percentage	Amount	When
1	true	false	(N/A)	(N/A)	(N/A)
2	false	true	3.5%	(N/A)	Anniversaries
3	true	true	(N/A)	\$50.00	Anniversaries
4	false	false	3.5%	(N/A)	OnceOnly
5	false	false	(N/A)	\$50.00	OnceOnly
6	true	true	3.5%	(N/A)	OnceOnly
7	true	true	(N/A)	\$50.00	OnceOnly
8	false	false	(N/A)	\$50.00	Anniversaries
9	false	true	(N/A)	(N/A)	(N/A)
10	true	false	3.5%	(N/A)	Anniversaries

Exclusions	Inclusions
	Description
	Accelerate
	RoundUp
	Percentage
	Amount
	When

**Figure 22 Test cases for “Other Payment Options” Covering Array Definition**

## Define the Main Covering Array Definition

With all the intricate conditions now neatly sorted out, the “Main” Covering Array Definition, which brings all the parameters together in a comprehensive test matrix, is simple to define: list all the remaining input fields, add test values, and link the two Covering Array Definition (see Figure 23).



**Figure 23 The “Main” Covering Array Definition bringing together all the inputs**

The set of test case inputs is listed below.

ID	Funding	IAD	FirstPymt	Accel	Round	%	Amount	When	Mortgage Amount	Product Category	Term	Promo Rate	Promo Period	Interes tRate	Am (months)	Pymt Frequency	PromoPymt	Pymt
1	(N/A)	2006-03-03	(N/A)	false	false	(N/A)	\$50.00	OnceOnly	\$100,000.00	Fixed	120	(N/A)	(N/A)	6.75	240	Weekly	(N/A)	(default)
2	2006-02-28	(N/A)	(N/A)	true	false	3.5%	(N/A)	Anniversaries	\$250,000.00	Fixed/2Rates/2Payments	36	0	9	5	300	Semimonthly	(default)	(default)
3	2006-02-24	(N/A)	(N/A)	true	true	(N/A)	\$50.00	OnceOnly	\$100,000.00	Fixed/2Rates/2Payments	60	4.55	9	6.75	300	Biweekly	\$1,400.54	(default)
4	(N/A)	(N/A)	2006-03-15	false	true	(N/A)	(N/A)	(N/A)	\$100,000.00	Variable	12	(N/A)	(N/A)	5	240	Monthly	(default)	(default)
5	2006-02-24	(N/A)	(N/A)	true	false	(N/A)	(N/A)	(N/A)	\$250,000.00	Fixed	84	(N/A)	(N/A)	5	300	Biweekly	(N/A)	(default)
6	2006-02-28	(N/A)	(N/A)	false	true	3.5%	(N/A)	Anniversaries	\$250,000.00	Variable	6	(N/A)	(N/A)	6.75	240	Semimonthly	(N/A)	(default)
7	(N/A)	2006-03-03	(N/A)	true	true	(N/A)	\$50.00	Anniversaries	\$250,000.00	Variable/2Rates/1Pymt	60	4.55	9	5	240	Weekly	(default)	(default)
8	(N/A)	(N/A)	2006-03-15	false	false	3.5%	(N/A)	OnceOnly	\$250,000.00	Fixed	36	(N/A)	(N/A)	6.75	300	Monthly	(N/A)	(default)
9	(N/A)	2006-03-03	(N/A)	true	true	3.5%	(N/A)	OnceOnly	\$100,000.00	Variable	6	(N/A)	(N/A)	5	300	Biweekly	(default)	(default)
10	2006-02-24	(N/A)	(N/A)	false	false	(N/A)	\$50.00	Anniversaries	\$250,000.00	Fixed	12	(N/A)	(N/A)	6.75	300	Semimonthly	(default)	(default)
11	(N/A)	2006-03-03	(N/A)	true	true	3.5%	(N/A)	OnceOnly	\$250,000.00	Fixed/2Rates/2Payments	84	0	9	6.75	240	Monthly	\$1,400.54	(default)
12	2006-02-28	(N/A)	(N/A)	false	false	3.5%	(N/A)	OnceOnly	\$100,000.00	Fixed/2Rates/2Payments	120	0	9	5	300	Weekly	\$1,400.54	(default)
13	(N/A)	2006-03-03	(N/A)	true	false	(N/A)	(N/A)	(N/A)	\$100,000.00	Variable/2Rates/1Pymt	60	0	9	6.75	300	Semimonthly	(N/A)	(default)
14	(N/A)	2006-03-03	(N/A)	false	true	3.5%	(N/A)	Anniversaries	\$100,000.00	Fixed/2Rates/2Payments	36	4.55	9	5	240	Biweekly	\$1,400.54	(default)
15	2006-02-28	(N/A)	(N/A)	true	true	(N/A)	\$50.00	Anniversaries	\$100,000.00	Fixed/2Rates/2Payments	84	4.55	9	6.75	300	Semimonthly	\$1,400.54	(default)
16	2006-02-28	(N/A)	(N/A)	false	false	(N/A)	\$50.00	OnceOnly	\$250,000.00	Fixed/2Rates/2Payments	120	4.55	9	5	300	Monthly	(default)	(default)
17	(N/A)	2006-03-03	(N/A)	true	true	(N/A)	\$50.00	OnceOnly	\$250,000.00	Variable	84	(N/A)	(N/A)	5	240	Weekly	(default)	(default)
18	(N/A)	2006-03-03	(N/A)	false	false	(N/A)	\$50.00	Anniversaries	\$100,000.00	Variable	60	(N/A)	(N/A)	5	240	Monthly	(N/A)	(default)
19	(N/A)	2006-03-03	(N/A)	false	true	(N/A)	(N/A)	(N/A)	\$250,000.00	Fixed/2Rates/2Payments	120	0	9	6.75	300	Biweekly	\$1,400.54	(default)
20	(N/A)	2006-03-03	(N/A)	true	false	3.5%	(N/A)	Anniversaries	\$100,000.00	Variable	12	(N/A)	(N/A)	6.75	240	Biweekly	(N/A)	(default)
21	(N/A)	(N/A)	2006-03-15	true	false	(N/A)	(N/A)	(N/A)	\$250,000.00	Fixed/2Rates/2Payments	120	4.55	9	6.75	240	Semimonthly	\$1,400.54	(default)
22	2006-02-24	(N/A)	(N/A)	true	true	(N/A)	\$50.00	Anniversaries	\$250,000.00	Fixed	6	(N/A)	(N/A)	5	240	Monthly	(N/A)	(default)
23	(N/A)	(N/A)	2006-03-15	false	true	3.5%	(N/A)	Anniversaries	\$100,000.00	Variable/2Rates/1Pymt	60	0	9	6.75	300	Biweekly	(default)	(default)
24	2006-02-28	(N/A)	(N/A)	false	false	(N/A)	\$50.00	Anniversaries	\$100,000.00	Variable/2Rates/1Pymt	60	4.55	9	6.75	240	Biweekly	(N/A)	(default)
25	(N/A)	2006-03-03	(N/A)	false	false	3.5%	(N/A)	OnceOnly	\$250,000.00	Variable	6	(N/A)	(N/A)	6.75	240	Weekly	(default)	(default)
26	2006-02-24	(N/A)	(N/A)	false	false	(N/A)	\$50.00	OnceOnly	\$250,000.00	Variable	36	(N/A)	(N/A)	5	240	Weekly	(default)	(default)
27	2006-02-28	(N/A)	(N/A)	true	true	3.5%	(N/A)	OnceOnly	\$100,000.00	Fixed	12	(N/A)	(N/A)	6.75	240	Weekly	(N/A)	(default)
28	2006-02-28	(N/A)	(N/A)	true	true	(N/A)	\$50.00	OnceOnly	\$250,000.00	Fixed	60	(N/A)	(N/A)	6.75	240	Semimonthly	(N/A)	(default)
29	2006-02-28	(N/A)	(N/A)	false	true	(N/A)	(N/A)	(N/A)	\$100,000.00	Variable/2Rates/1Pymt	60	4.55	9	5	240	Weekly	(N/A)	(default)
30	2006-02-24	(N/A)	(N/A)	true	false	3.5%	(N/A)	Anniversaries	\$100,000.00	Variable/2Rates/1Pymt	60	4.55	9	5	240	Monthly	(default)	(default)
31	(N/A)	(N/A)	2006-03-15	true	false	(N/A)	(N/A)	(N/A)	\$100,000.00	Variable	6	(N/A)	(N/A)	6.75	300	Weekly	(default)	(default)



ID	Funding	IAD	FirstPymt	Accel	Round	%	Amount	When	Mortgage Amount	Product Category	Term	Promo Rate	Promo Period	InterestRate	Am (months)	Pymt Frequency	PromoPymt	Pymt
32	(N/A)	(N/A)	2006-03-15	false	false	(N/A)	\$50.00	OnceOnly	\$100,000.00	Fixed/2Rates/2Payments	84	0	9	5	240	Semimonthly	\$1,400.54	(default)
33	2006-02-28	(N/A)	(N/A)	true	false	(N/A)	(N/A)	(N/A)	\$250,000.00	Fixed	12	(N/A)	(N/A)	5	300	Monthly	(N/A)	(default)
34	(N/A)	(N/A)	2006-03-15	false	false	(N/A)	\$50.00	Anniversaries	\$250,000.00	Fixed/2Rates/2Payments	120	0	9	5	240	Weekly	\$1,400.54	(default)
35	(N/A)	(N/A)	2006-03-15	false	false	(N/A)	\$50.00	OnceOnly	\$100,000.00	Variable	6	(N/A)	(N/A)	5	240	Biweekly	(N/A)	(default)
36	2006-02-24	(N/A)	(N/A)	false	true	3.5%	(N/A)	Anniversaries	\$100,000.00	Fixed	12	(N/A)	(N/A)	6.75	300	Monthly	(N/A)	(default)
37	2006-02-24	(N/A)	(N/A)	false	false	3.5%	(N/A)	OnceOnly	\$100,000.00	Variable	120	(N/A)	(N/A)	6.75	240	Semimonthly	(default)	(default)
38	2006-02-24	(N/A)	(N/A)	true	true	3.5%	(N/A)	OnceOnly	\$250,000.00	Variable/2Rates/1Pymt	60	0	9	5	300	Semimonthly	(N/A)	(default)
39	(N/A)	(N/A)	2006-03-15	true	true	(N/A)	\$50.00	OnceOnly	\$250,000.00	Fixed/2Rates/2Payments	120	0	9	6.75	300	Monthly	(default)	(default)
40	(N/A)	(N/A)	2006-03-15	true	true	(N/A)	\$50.00	OnceOnly	\$100,000.00	Fixed	6	(N/A)	(N/A)	5	300	Biweekly	(N/A)	(default)
41	(N/A)	(N/A)	2006-03-15	true	true	(N/A)	\$50.00	Anniversaries	\$100,000.00	Variable	12	(N/A)	(N/A)	6.75	300	Biweekly	(default)	(default)
42	(N/A)	(N/A)	2006-03-15	true	false	3.5%	(N/A)	Anniversaries	\$250,000.00	Fixed	84	(N/A)	(N/A)	6.75	300	Weekly	(N/A)	(default)
43	(N/A)	(N/A)	2006-03-15	false	false	(N/A)	\$50.00	Anniversaries	\$100,000.00	Variable	6	(N/A)	(N/A)	5	300	Weekly	(N/A)	(default)
44	2006-02-24	(N/A)	(N/A)	false	false	3.5%	(N/A)	OnceOnly	\$250,000.00	Variable	12	(N/A)	(N/A)	6.75	240	Biweekly	(default)	(default)
45	2006-02-24	(N/A)	(N/A)	false	true	(N/A)	(N/A)	(N/A)	\$250,000.00	Fixed	6	(N/A)	(N/A)	5	300	Semimonthly	(default)	(default)
46	2006-02-24	(N/A)	(N/A)	false	false	(N/A)	\$50.00	OnceOnly	\$250,000.00	Variable	12	(N/A)	(N/A)	6.75	240	Weekly	(N/A)	(default)
47	(N/A)	(N/A)	2006-03-15	true	false	(N/A)	(N/A)	(N/A)	\$250,000.00	Fixed/2Rates/2Payments	36	0	9	6.75	240	Monthly	(default)	(default)
48	2006-02-24	(N/A)	(N/A)	true	false	3.5%	(N/A)	Anniversaries	\$250,000.00	Fixed	6	(N/A)	(N/A)	6.75	300	Biweekly	(N/A)	(default)
49	2006-02-28	(N/A)	(N/A)	true	true	(N/A)	\$50.00	OnceOnly	\$100,000.00	Variable	12	(N/A)	(N/A)	6.75	240	Weekly	(default)	(default)
50	(N/A)	(N/A)	2006-03-15	true	true	(N/A)	\$50.00	Anniversaries	\$100,000.00	Fixed/2Rates/2Payments	36	0	9	5	240	Weekly	\$1,400.54	(default)
51	2006-02-28	(N/A)	(N/A)	false	false	3.5%	(N/A)	OnceOnly	\$250,000.00	Variable/2Rates/1Pymt	60	4.55	9	6.75	300	Weekly	(default)	(default)
52	2006-02-28	(N/A)	(N/A)	true	false	3.5%	(N/A)	Anniversaries	\$100,000.00	Fixed/2Rates/2Payments	120	0	9	5	300	Biweekly	\$1,400.54	(default)
53	(N/A)	(N/A)	2006-03-15	true	true	3.5%	(N/A)	OnceOnly	\$100,000.00	Fixed/2Rates/2Payments	120	4.55	9	6.75	300	Biweekly	(default)	(default)
54	(N/A)	(N/A)	2006-03-15	true	true	3.5%	(N/A)	OnceOnly	\$250,000.00	Fixed/2Rates/2Payments	36	4.55	9	5	240	Weekly	\$1,400.54	(default)
55	2006-02-28	(N/A)	(N/A)	false	true	3.5%	(N/A)	Anniversaries	\$250,000.00	Variable	84	(N/A)	(N/A)	5	240	Weekly	(default)	(default)
56	(N/A)	(N/A)	2006-03-15	true	true	(N/A)	\$50.00	OnceOnly	\$100,000.00	Variable	36	(N/A)	(N/A)	5	300	Monthly	(default)	(default)
57	2006-02-28	(N/A)	(N/A)	false	false	3.5%	(N/A)	OnceOnly	\$250,000.00	Variable	84	(N/A)	(N/A)	6.75	240	Semimonthly	(default)	(default)
58	2006-02-24	(N/A)	(N/A)	false	false	(N/A)	\$50.00	OnceOnly	\$100,000.00	Variable/2Rates/1Pymt	60	4.55	9	5	240	Semimonthly	(default)	(default)
59	(N/A)	(N/A)	2006-03-15	false	false	(N/A)	\$50.00	Anniversaries	\$100,000.00	Variable	36	(N/A)	(N/A)	5	300	Weekly	(N/A)	(default)
60	(N/A)	(N/A)	2006-03-15	false	true	(N/A)	(N/A)	(N/A)	\$250,000.00	Variable	36	(N/A)	(N/A)	6.75	240	Weekly	(default)	(default)
61	(N/A)	2006-03-03	(N/A)	false	false	(N/A)	\$50.00	Anniversaries	\$100,000.00	Fixed	84	(N/A)	(N/A)	6.75	300	Biweekly	(N/A)	(default)
62	(N/A)	(N/A)	2006-03-15	false	true	3.5%	(N/A)	Anniversaries	\$250,000.00	Variable	120	(N/A)	(N/A)	6.75	300	Biweekly	(N/A)	(default)

ID	Funding	IAD	FirstPymt	Accel	Round	%	Amount	When	Mortgage Amount	Product Category	Term	Promo Rate	Promo Period	InterestRate	Am (months)	Pymt Frequency	PromoPymt	Pymt
63	(N/A)	(N/A)	2006-03-15	true	true	(N/A)	\$50.00	Anniversaries	\$250,000.00	Variable	120	(N/A)	(N/A)	6.75	300	Biweekly	(default)	(default)
64	2006-02-28	(N/A)	(N/A)	true	true	(N/A)	\$50.00	OnceOnly	\$250,000.00	Variable/2Rates/1Pymt	60	0	9	5	300	Monthly	(N/A)	(default)
65	(N/A)	2006-03-03	(N/A)	false	true	(N/A)	(N/A)	(N/A)	\$250,000.00	Fixed/2Rates/2Payments	84	4.55	9	6.75	300	Weekly	\$1,400.54	(default)

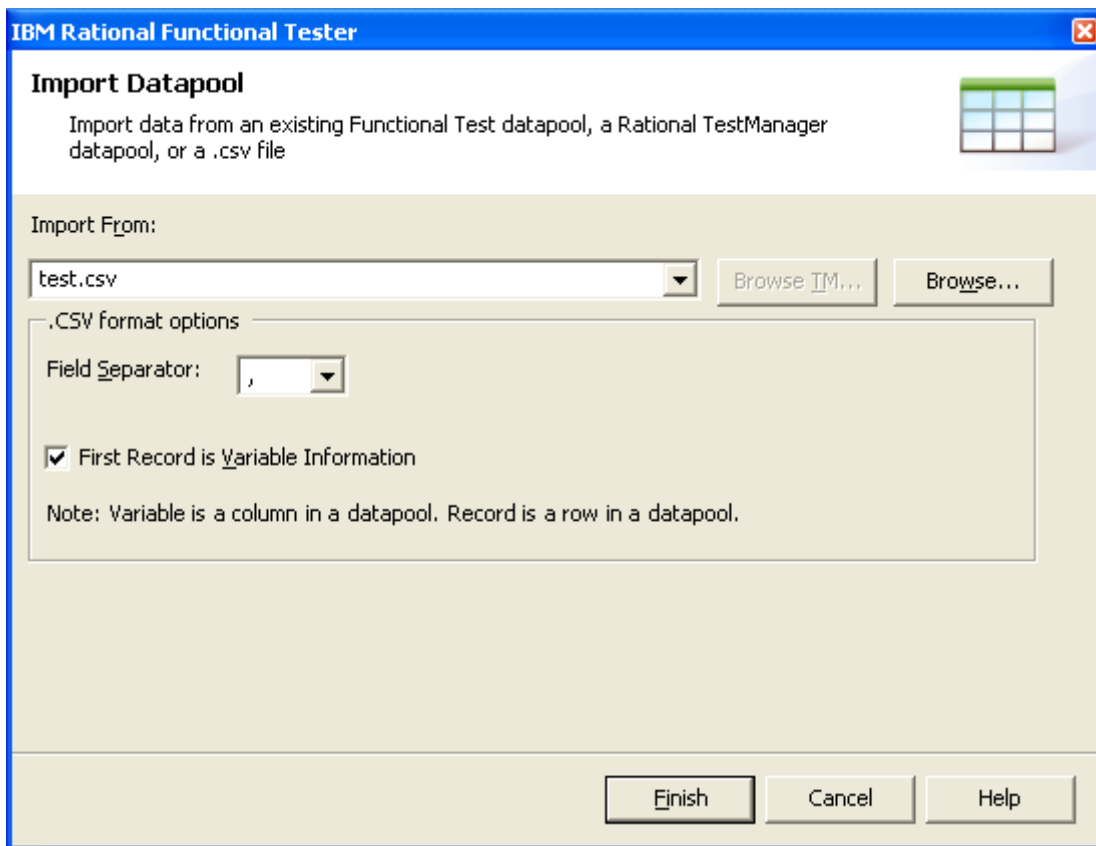
Table 13 Test Cases generated by the "Main" Covering Array Definition

## Integration with IBM Rational Functional Tester

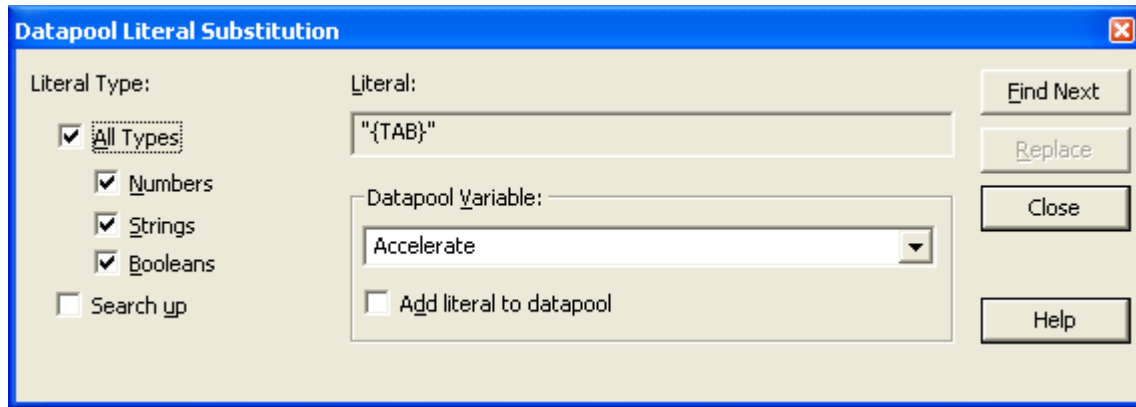
**Note:** The information in this section is applicable for QTAssistant MLCA plug-in version 1.0 and IBM Rational Functional Tester 6.1 (RFT). For up-to-date information on QTAssistant's MLCA plug-in integration with other automation tools, please visit <<http://www.qtassistant.com/mlca>>.

The easiest way to integrate QTAssistant MLCA plug-in with IBM Rational Functional Tester 6.1 is through Datapools.

1. In QTAssistant, implement the model. Upon completion, export the test cases to file using the comma separated values format (extension \*.csv).
2. In RFT, develop the automation script.
3. In RFT, create a new Datapool. Import the file you have exported from QTAssistant. Make sure that "First Record is Variable Information" checkbox is ticked.



4. Upon successful completion, the information imported from the file should be displayed in a data grid.
5. Associate the Datapool you've just created with the script you're working with.
  - a. In RFT, from the Script menu, click "Find Literals and Replace with Datapool References..."



- b. From the dropdown, select the “Datapool Variable” to work with. Click “Find Next”.
  - c. In the script, replace the appropriate literal(s).
6. Repeat 5.a, 5.b and 5.c as needed.

## References

- [CA04] Colbourn, Charles J. *Combinatorial Aspects of Covering Arrays*, Computer Science and Engineering Arizona State University, August 2004
- [TP05] OMG *UML Testing Profile Version 1.0*, July 2005
- [NS02] National Institute of Standards and Technology *The Economic Impacts of Inadequate Infrastructure for Software Testing*, U.S. Department of Commerce, May 2002
- [DE02] Kuhn, D. Richard, Reilly, Michael J. *An Investigation of the Applicability of Design of Experiments to Software Testing*, National Institute of Standards and Technology, December 2002