A Human-centred Framework for Combinatorial Test Design

Maria Spichkova\textsuperscript{1} and Anna Zamansky\textsuperscript{2}

\textsuperscript{1}School of Science, RMIT University, 414-418 Swanston Street, 3001, Melbourne, Australia
\textsuperscript{2}Information Systems Department, University of Haifa, Carmel Mountain, 31905, Haifa, Israel

Keywords: Software Quality, Testing, Formal Methods, Combinatorial Test Design.

Abstract: This paper presents AHR, a formal framework for combinatorial test design that is Agile, Human-centred and Refinement-oriented. The framework (i) allows us to reuse test plans developed for an abstract level at more concrete levels; (ii) has human-centric interface providing queries and alerts whenever the specified test plan is incomplete or invalid; (iii) involves analysis of the testing constraints within combinatorial testing.

1 INTRODUCTION

Combinatorial Test Design (CTD) is an effective methodology for test design of complex software systems. In CTD systems are modelled via a set of parameters, their respective values and restrictions on the value combinations, cf. (Nie and Leung, 2011; Zhang et al., 2014). The main challenge of CTD is to optimise the number of test cases, while ensuring the coverage of given conditions. One of the most standard coverage requirements is pairwise testing (Nie and Leung, 2011), where every (executable) pair of possible values of system parameters is considered. Experimental work shows that using tests sets with exhaustive covering of a small number of parameters (such as pairwise testing) can typically detect more than 50-75\% of the bugs in a program, cf. (Tai and Lei, 2002; Kuhn et al., 2004). This testing approach can be applied at different phases and scopes of testing, including end-to-end and system-level testing and feature-, service- and application program interface-level testing.

The CTD approach is model-based, i.e., test plans are derived (manually or automatically) on the basis from a model of the system under test (SUT) and its environment. Therefore, while using this approach, a considerable time have to be spend on generating the infrastructure for testing (including the model of SUT) instead of hand-crafting individual tests. This implies that only behaviour encoded in the model can be tested.

Moreover, in many cases different behaviours need to be tested at different stages of the development cycle. This leads to the need for handling multiple abstraction levels and a systematic way of bridging between them. However, to provide an adequate model with an sufficient abstraction remains a strictly human activity, which heavily relies on the human factor. One barrier in the adoption of MBT in industry is the steep learning curve for modelling notations, cf. (Grieskamp, 2006). Another barrier is the lack of state-of-the-art authoring environments.

In this work, we aim is to provide the corresponding semi-automatic support for the tester and help minimise the number of human errors as well as their impact on the system-under-test.

We propose AHR, a formal framework for the construction of combinatorial models within multiple levels of abstraction. The main idea is defining explicit refinement relations between elements of the model at different abstraction levels. This core features of our framework are (i) the reuse of test plans developed for an abstract level at more concrete levels; (ii) human-centric interface providing queries and alerts to testers to help them analyse whenever the specified test plans, model and/or constraints are incomplete or invalid. One of the AHR goals is to provide a sufficient support to the tester by semi-automatic analysis of the model and the test plans.

Outline: The rest of the paper is organised as follows. In Section 2 we discuss the related work and the corresponding motivation for the AHR development. Section 3 presents the background on CTD. Section 4 provides the formal definitions that build the core of AHR to support CTD within multiple abstraction levels. In Section 5 we discuss a use case for the framework application. In Section 6 we summarise the paper and propose directions for future research.
2 RELATED WORK

The advantage of MBT is that the testers can concentrate on system model and constraints instead of the manual specification of individual tests. There are many approaches on model-based testing, e.g., (Dalal et al., 1999; Grieskamp, 2006). Utting et al. presented a taxonomy of MBT approaches in (Utting et al., 2012). There are also many approaches on CTD, cf. (Zhang et al., 2014; Segall et al., 2012; Farchi et al., 2014; Farchi et al., 2013; Kuhn et al., 2011). However, most of them focus on the question how to generate test cases from a model in the most efficient way also achieving full coverage of the required system properties by the generated test cases. In our approach, we combine the ideas of CTD with the idea of a step-wise refinement of the system through the development process, also following agile modelling practices and guidelines (Hellmann et al., 2012; Talby et al., 2006). Agile software development process focuses on facilitating early and fast production of working code (Turk et al., 2005; Rumpe, 2006; Hazzan and Dubinsky, 2014) by supporting iterative, incremental development, where with each iteration we refine the system step by step.

As pointed out in (Pretschner, 2005), model-based testing makes sense only if the model is more abstract than SUT. Testing methodologies for complex systems often integrate different abstraction levels of the system representation (Broy, 2005; Spichkova, 2008). Thus, abstraction plays a key role in the process of system modelling. An important domain in which modelling with different levels of abstraction is particularly beneficiary is cyber-physical systems (CPSSs). Several works proposed to use a platform-independent architectural design in the early stages of system development, while pushing hardware- and software-dependent design as late as possible (Sapienza et al., 2012; Spichkova and Campetelli, 2012; Blech et al., 2014). In our previous work (Spichkova et al., 2015a) we suggested to use three main meta-levels of abstraction for the CPS development: abstract, virtual, and cyber-physical. The AHR framework can be applied at any of these meta-levels.

In (Segall and Tzoref-Brill, 2012) a tool for supporting interactive refinement of combinatorial test plans by the tester was presented. This tool is meant for manual modifications of existing test plans, is align with the idea of Human-Centred Agile Test Design (Zamansky and Farchi, 2015; Spichkova et al., 2015b) where it is explicitly acknowledged that the tester’s activity is not error-proof. This tool be a good support for the tester, but it does not cover the following point that we consider as crucial for development of complex systems: refinement-based development, where the tester is working at multiple abstraction levels. We aim to cover this point in the proposed AHR framework: If we trace the refinement relations not only between the properties but also between test plans, this might also help to correct possible mistakes more efficiently, as well as provide additional support if the system model is modified.

3 CTD: FORMAL BACKGROUND

In CTD a system is modelled using a finite set of system parameters \( \mathcal{A} = \{A_1, \ldots, A_n\} \). To each of the parameters is associated a set of corresponding values \( \mathcal{V} = \{\mathcal{V}(A_1), \ldots, \mathcal{V}(A_n)\} \).

In what follows we use the notion of interaction between the different values of the parameters and the notion of test coverage:

Definition 1. An interaction for a set of system parameters \( \mathcal{A} \) is an element of the form \( I \subseteq \bigcup_{i=1}^{n} \mathcal{V}(A_i) \), where at most one value of each parameter \( A_i \) may appear.

Definition 2. A test (or scenario) is an interaction of size \( n \), where \( n \) is the number of system parameters.

Definition 3. A set of tests \( \mathcal{T} \) covers a set of interactions \( \mathcal{C} \) (denoted by \( \mathcal{T} \models \mathcal{C} \)) if for every \( c \in \mathcal{C} \) there is some \( t \in \mathcal{T} \), such that \( c \subseteq t \).

Definition 4. A combinatorial model \( \mathcal{E} \) of a system with the corresponding set of parameters \( \mathcal{A} \) is a set of tests, which defines all tests over \( \mathcal{A} \) that are executable in the system.

Definition 5. A test plan is a triple \( \text{Plan} = (\mathcal{E}, C, \mathcal{T}) \), where \( \mathcal{E} \) is a combinatorial model, \( C \) is a set of interactions called coverage requirements, \( \mathcal{T} \) is a set tests, and the relation \( \mathcal{T} \models \mathcal{C} \) holds.

In the above terms, a pairwise test plan can be specified as any pair of the form

\[
\text{Plan} = (\mathcal{E}, C_{\text{pair}}(\mathcal{E}), \mathcal{T})
\]

where \( C_{\text{pair}} \) is the set of all interactions of size 2 which can be extended to scenarios from \( \mathcal{E} \).

Example 1. For a running example scenario, let us consider a cyber-physical system with two robots \( R_1 \) and \( R_2 \) that are interacting with each other. At some level of abstraction (let us call it Level1), a robot can be modelled by two parameters, \( GM \) and \( P \). Thus,

\[ \mathcal{A} = \{GM_1, GM_2, P_1, P_2\} \]

The system parameters \( GM_1 \) and \( GM_2 \) specify the gripper modes (which can be either closed to hold an
object or open) of robots $R_1$ and $R_2$ respectively. Let us consider that at this level of abstraction the gripper have only two modes:

\[ \forall (GM_1) = \forall (GM_2) = \{\text{open, closed}\} \]

$P_1$ and $P_2$ represent the robots’ positions. We assume at this level of abstraction that the grippers of each robot have only three possible positions:

\[ \forall (P_1) = \forall (P_2) = \{\text{pos1, pos2, pos3}\} \]

In what follows let us assume pairwise coverage requirements. We now specify a meta-operation $\text{Give}(A, B)$ to model the scenario when the robot $A$ hands an object to the robot $B$. $\text{Give}(A, B)$ can only be performed when the grippers of both robots are in the same position, the gripper of $A$ is closed and the gripper of $B$ is open (where $A, B \in \{R_1, R_2\}$ and $A \neq B$). Thus, the operation $\text{Give}(R_1, R_2)$ can be captured on Level1 in the following constraint model $M_{\text{Give}(R_1, R_2)}$:

\[ P_1 = P_2 \land GM_1 = \text{closed} \land GM_2 = \text{open} \]

Without any constraints, we would require 36 tests to cover all possible combinations of the values, but considering the full coverage of the $M_{\text{Give}(R_1, R_2)}$, we require three tests only, cf. Table 1.

At the next level of abstraction, Level2, we might refine both $\mathcal{A}$ and $\mathcal{V}$ to obtain a more realistic model of the system. In the next section, we introduce the notion of parameter and value refinements, which provides an explicit specification of the relations between abstraction levels to allow traceability of the model modification and the corresponding test sets.

Table 1: Test set providing pairwise coverage for $\text{Give}(R_1, R_2)$ on Level1.

<table>
<thead>
<tr>
<th>testID</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$GM_1$</th>
<th>$GM_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>test1</td>
<td>pos1</td>
<td>pos1</td>
<td>closed</td>
<td>open</td>
</tr>
<tr>
<td>test2</td>
<td>pos2</td>
<td>pos2</td>
<td>closed</td>
<td>open</td>
</tr>
<tr>
<td>test3</td>
<td>pos3</td>
<td>pos3</td>
<td>closed</td>
<td>open</td>
</tr>
</tbody>
</table>

4 REFINEMENT-BASED DEVELOPMENT

Our framework is based on the idea of refinement: a more concrete model can be substituted for an abstract one as long as its behaviour is consistent with that defined in the abstract model.

Definition 6. Let us consider two sets of system parameters $\mathcal{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ and $\mathcal{B} = \{\mathcal{B}_1, \ldots, \mathcal{B}_k\}$, with $k \geq n$. We define a parameter refinement from $\mathcal{A}$ to $\mathcal{B}$ (also denoted by $\mathcal{A} \leadsto \mathcal{B}$) as a function $R$ that maps each parameter $\mathcal{A}_i$ to a set of parameters from $\mathcal{B}$, so that for two distinct parameters $\mathcal{A}_i$ and $\mathcal{A}_j$, $1 \leq i, j \leq n, i \neq j$, the sets $R(\mathcal{A}_i)$ and $R(\mathcal{A}_j)$ are disjoint.

Definition 7. For a parameter refinement $\mathcal{R} : \mathcal{A} \leadsto \mathcal{B}$, a value refinement $\forall \mathcal{R} : \forall (\mathcal{A}) \leadsto \forall (\mathcal{B})$ maps each value $v \in \forall (\mathcal{A})$ to the corresponding set of values $\forall \mathcal{R}(v)$, where

\[ \forall \mathcal{R}(v) \subseteq \bigcup_{B \in \mathcal{R}(\mathcal{A})} \forall (\mathcal{B}) \]

such that if $B_j \in \mathcal{R}(\mathcal{A}_i)$, then for every $v \in \forall (\mathcal{A}_i)$, $\forall (B_j) \cap \forall \mathcal{R}(v) \neq \emptyset$.

The above definitions do not exclude the case where both $\mathcal{R}$ and $\forall \mathcal{R}$ are singleton functions, i.e. functions that convert each element $a$ to a singleton $\{a\}$. For this reason we have to introduce the notion of concretisation.

Definition 8. If there exist parameter refinement $\mathcal{R}$ and value refinement $\forall \mathcal{R}$ from a set of system parameters $\mathcal{A}$ to a set of system parameters $\mathcal{B}$ (where at least one of the functions $\mathcal{R}$ and $\forall \mathcal{R}$ is not a singleton function), we say that $\mathcal{B}$ is a concretisation (strict refinement) of $\mathcal{A}$ with respect to $\mathcal{R}$ and $\forall$. We denote this by $\mathcal{A} \models R$.

Example 2. Let us continue with the running example of two interacting robots. At Level1, we have the set of system parameters $\mathcal{A}^{level1} = \{P_1, P_2, GM_1, GM_2\}$. At Level2, we refine the $\forall (GM_1)$ and $\forall (GM_2)$ to have an additional the gripper mode $mid$, representing an intermediate position between open and closed (i.e., the position when the grippers are opening or closing, but not yet completely open or closed). We do not need to change the parameters $GM_1$ and $GM_2$, but we have to extend the sets $\forall (GM_1)$ and $\forall (GM_2)$.

We also refine the abstract positions to their two-dimensional coordinates: for $i \in [1, 2]$, $P_i$ is refined to the tuple of two new parameters $X_i$ and $Y_i$, and the elements of $\forall (P_i)$ are mapped to the tuples of the corresponding coordinates. Thus, at Level2, we have

\[ \mathcal{A}^{level2} = \{X_1, Y_1, X_2, Y_2, GM_1, GM_2\} \]

\[ \forall (X_1) = \forall (X_2) = \{x_1, x_2\} \]

\[ \forall (Y_1) = \forall (Y_2) = \{y_1, y_2, y_3\} \]

\[ \forall (GM_1) = \forall (GM_2) = \{\text{open, closed, mid}\} \]

To represent the concretisation from Level1 to Level2, we specify the following relations for $i \in [1, 2]$ (cf. also Figures 1 and 2):

(1) Parameter refinement $GM^{level1}_i \leadsto GM^{level2}_i$ is a singleton function. The corresponding value refinements are
Figure 1: Parameter Refinement.

\[ VR(open) = \{open\} \] and \[ VR(closed) = \{closed\} \]
where mid \( \in \mathcal{V}(GM^{Level_2}) \) does not have any corresponding element on Level_1.

(2) Parameter refinement \( P_i \mapsto (X_i, Y_i) \) maps an abstract position to a tuple of two-dimensional coordinates, where the corresponding value refinements are:
- \( \mathcal{V}_R(pos_1) = \{(x_1, y_1), (x_2, y_1)\} \)
- \( \mathcal{V}_R(pos_2) = \{(x_1, y_2), (x_2, y_2)\} \)
- \( \mathcal{V}_R(pos_3) = \{(x_1, y_3), (x_2, y_3)\} \)

Definition 9. A model refinement \( M_R \) is a mapping from the elements (conjuncts) of the constraint model \( M' \) specified on the abstraction level \( i \) over the set of parameters \( \mathcal{A} \) to the constraint model \( M'^{+1} \) specified on the next abstraction level over the set of parameters \( \mathcal{B} \), where \( \mathcal{A} \Rightarrow \mathcal{B} \).

Definition 10. A test refinement \( T_R \) is a mapping from the set of tests over \( \mathcal{A} \) to the set of tests over \( \mathcal{B} \), where \( \mathcal{A} \Rightarrow \mathcal{B} \) and

\[ R : \mathcal{A} \mapsto \mathcal{B} \] is a parameter refinement with the corresponding value refinement \( VR : \mathcal{V}(A) \mapsto \mathcal{V}(B) \).

The above provides a theoretical basis for tester support: given a system model based on a set of parameters \( \mathcal{A} \), the tester can specify explicit parameter and value refinements, which in its turn induces a system model for \( \mathcal{B} \) and the refinement relations between sets of tests on different abstract levels.

5 USE CASE FOR TESTER SUPPORT

Suppose the modeller already has constructed a model at Level_1, using the parameters from our running example on the Give meta-operation and providing the constraint model

\[ GM_1 = closed \land GM_2 = open \] (2)

where the information on the position is erroneously omitted, because of a human error. If we generate tests automatically, we obtain 9 tests to cover the model, cf. Table 2. Let us consider that the tester decided to limit the test set to have two tests only, e.g., \( \{P_1 : pos_1, P_2 : pos_2, GM_1 : closed, GM_2 : open\} \) and \( \{P_1 : pos_2, P_2 : pos_2, GM_1 : closed, GM_2 : open\} \).

The proposed framework would analyse these tests to come up with the corresponding logical constraint:

\[ GM_1 = closed \land GM_2 = open \land (P_1 = P_2 = pos_1 \lor P_1 = P_2 = pos_2) \] (3)

Table 2: Test set providing pairwise coverage for \( \text{Give}(R_1, R_2) \) under constraint (2).

<table>
<thead>
<tr>
<th>testID</th>
<th>( P_1 )</th>
<th>( P_2 )</th>
<th>( GM_1 )</th>
<th>( GM_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>test1</td>
<td>pos_1</td>
<td>pos_1</td>
<td>closed</td>
<td>open</td>
</tr>
<tr>
<td>test2</td>
<td>pos_1</td>
<td>pos_2</td>
<td>closed</td>
<td>open</td>
</tr>
<tr>
<td>test3</td>
<td>pos_1</td>
<td>pos_2</td>
<td>closed</td>
<td>open</td>
</tr>
<tr>
<td>test4</td>
<td>pos_2</td>
<td>pos_1</td>
<td>closed</td>
<td>open</td>
</tr>
<tr>
<td>test5</td>
<td>pos_2</td>
<td>pos_2</td>
<td>closed</td>
<td>open</td>
</tr>
<tr>
<td>test6</td>
<td>pos_2</td>
<td>pos_3</td>
<td>closed</td>
<td>open</td>
</tr>
<tr>
<td>test7</td>
<td>pos_3</td>
<td>pos_3</td>
<td>closed</td>
<td>open</td>
</tr>
<tr>
<td>test8</td>
<td>pos_3</td>
<td>pos_2</td>
<td>closed</td>
<td>open</td>
</tr>
</tbody>
</table>

The AHR framework checks whether the coverage is achieved by the above two tests, and provide the corresponding alert to the tester along with the message that the constraint models (2) and (3) are semantically unequal, (3) is a stronger constraint than (2). Let us consider that the tester changes the constraint model to (3) and select an additional test \( \{P_1 : pos_3, P_2 : pos_3, GM_1 : closed, GM_2 : open\} \).
Next, the system model is refined as presented in Example 2. Based on the specification of the system parameters concretisation, the framework provides the following suggestion for the refinement of the constraint model $M_{\text{Give}(R_1,R_2)}$. To increase the readability and the traceability of the refinement steps, we suggest the provision in two forms: as the constructed constraint model, cf. (4) and as a mapping from the models on the previous and the current abstraction levels, cf. Table 3.

\[
X_1 = X_2 \land Y_1 = Y_2 \land \\
GM_1 = \text{closed} \land GM_2 = \text{open} 
\]  

(4)

Table 3: Model refinement for $\text{Give}(R_1,R_2)$.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1 = P_2$</td>
<td>$X_1 = X_2 \land Y_1 = Y_2$</td>
</tr>
<tr>
<td>$GM_1 = \text{closed}$</td>
<td>$GM_1 = \text{closed}$</td>
</tr>
<tr>
<td>$GM_2 = \text{open}$</td>
<td>$GM_2 = \text{open}$</td>
</tr>
</tbody>
</table>

Depending on the semantics we give to the spatial constrains in our model, we accept this suggestion or adapt it. If we assume that the robot $R_1$ can give an object to the robot $R_2$ when their grippers have the same abstract coordinates, we accept this suggestion and the framework proceeds with the refinement of the tests. However, we might also assume at Level 2 that the robots’ grippers cannot have the same coordinates (except a collision situation), and that the robot $R_1$ can give an object to the robot $R_2$ when their grippers are at the same level, but their x-coordinates have to be different. In this case the constraint model has to be specified on Level 2 as presented by (5) and Table 4.

\[
X_1 \neq X_2 \land Y_1 = Y_2 \land \\
GM_1 = \text{closed} \land GM_2 = \text{open} 
\]  

(5)

Table 4: Corrected model refinement for $\text{Give}(R_1,R_2)$.

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1 = P_2$</td>
<td>$X_1 \neq X_2 \land Y_1 = Y_2$</td>
</tr>
<tr>
<td>$GM_1 = \text{closed}$</td>
<td>$GM_1 = \text{closed}$</td>
</tr>
<tr>
<td>$GM_2 = \text{open}$</td>
<td>$GM_2 = \text{open}$</td>
</tr>
</tbody>
</table>

For the corrected model (5), AHR generates 6 tests to achieve the coverage (cf. Table 5), and suggest the following mapping between sets of tests:

- $test_1^{\text{Level}} \leadsto \{test_1^{\text{Level}}, test_2^{\text{Level}}\}$
- $test_2^{\text{Level}} \leadsto \{test_3^{\text{Level}}, test_4^{\text{Level}}\}$
- $test_3^{\text{Level}} \leadsto \{test_5^{\text{Level}}, test_6^{\text{Level}}\}$

Traceability not only between the modification in the system parameters but also between constraint models and between test plans, helps to correct possible mistakes more efficiently, as well as provides additional support if the system model is modified. For example, if on some stage a new constraint is identified that the meta-operation $\text{Give}(R_1,R_2)$ is not possible when the robots’ grippers are in the position pos2, the required changes in the models and the corresponding test plans for all concretisations of the model can be easily identified. Moreover, the AHR framework also allows analysis of several branches of the refinement.

### 6 CONCLUSIONS

This paper presents our ongoing work on human-centred testing. We propose a formal framework for combinatorial test design that is Agile, Human-centred and Refinement-oriented. The framework

- allows us to reuse test plans developed for an abstract level at more concrete levels;

- involves analysis of the testing constraints.

We integrate the ideas of refinement-based development and the agile CTD, aim at increasing of the readability and understandability of tests, to conform with the ideas of human-oriented software development, cf. (Spichkova et al., 2013; Spichkova, 2013).

A further future work direction is an implementation of a tool prototype for the proposed framework. To this end we plan to connect the prototype with the environment of IBM Functional Coverage Unified Solution, cf. (Segall and Tzoref-Brill, 2012; Wojciak and Tzoref-Brill, 2014), which is a tool for test-oriented system modelling, focused on model based test planning and functional coverage analysis.

---

1The second author was supported by The Israel Science Foundation under grant agreement no. 817/15.
REFERENCES


A Human-centred Framework for Combinatorial Test Design

233