Building a tool for generating test frames automatically based on Category Partition method and applying it to the organization unit management module of DHIS2 software.

Thanh Ngoc Nguyen
University of Oslo
thanhng@ifi.uio.no

ABSTRACT
This project is to develop a tool to generate automatically test frames from the test specification based on the category partition method. This tool has been applied to a real application, a module of open source software for health data collection and processing.

KEYWORDS: black box testing, category partition method, dhis2, testing
# Table of Contents

ABSTRACT ........................................................................................................................................................................... 1

KEYWORDS: black box testing, category partition method, dhis2, testing .............................................................................. 1

1 introduction ........................................................................................................................................................................... 5

1.1 Scope of project ........................................................................................................................................................... 5

1.2 Goals of project ............................................................................................................................................................ 5

1.3 Structure of the report .................................................................................................................................................. 5

2 BACKGROUND ............................................................................................................................................................... 5

2.1 Why black box? ............................................................................................................................................................ 5

2.2 Why category partition? ............................................................................................................................................... 6

2.3 Why test frame generator? ......................................................................................................................................... 6

3 DESIGN OF THE CASE STUDY .................................................................................................................................. 7

3.1 Description of system under test (SUT) ..................................................................................................................... 7

3.2 The dhis-web-maintainance-organisationunit module ............................................................................................... 7

3.3 The requirement specification of the module ........................................................................................................... 9

3.4 Building an automation tool for generating test frames based on test specification using CPM ........................... 12

3.4.1 Standardization of test specification language (TSL) ........................................................................................... 12

3.4.2 A revised TSL ......................................................................................................................................................... 12

3.4.3 Using Excel file as input/output .......................................................................................................................... 13

3.4.4 Building algorithm .................................................................................................................................................. 13

3.5 Initial results ................................................................................................................................................................. 13

3.5.1 Organization unit functionalities ........................................................................................................................... 14

3.5.2 Organization unit group ...................................................................................................................................... 19

3.5.3 Group set .............................................................................................................................................................. 22

3.5.4 Hierarchy operators ............................................................................................................................................. 26

4 ANALYSIS OF THE RESULTS .................................................................................................................................... 27

4.1 Analyzing the results .................................................................................................................................................... 27

4.1.1 Effectiveness of CPM: ........................................................................................................................................... 27

4.1.2 Effectiveness of the tool: ...................................................................................................................................... 27

4.1.3 Results of running the test set: ............................................................................................................................ 29

4.1.4 The relations between graphical user interface (GUI) design and CPM: ............................................................ 29

4.2 Summary of important results .................................................................................................................................. 30

5 LESSONS LEARNED AND OPEN ISSUES .................................................................................................................. 30
5.1  Lesson learned............................................................................................................................................................. 30
5.2  Practical and technical difficulties........................................................................................................................................ 30
5.3  Discussion, limitation, and future research.................................................................................................................. 31
6   REFERENCES................................................................................................................................................................. 32
7   APPENDICES.................................................................................................................................................................. 34
7.1  The source code of the tool developed in the project .................................................................................................. 34
7.2  Source code being tested ............................................................................................................................................. 41
List of Tables

Table 1: Code layout of the SUT ................................................................................................................................................. 8
Table 2: System specification .................................................................................................................................................... 12
Table 3: Simplified TSL ............................................................................................................................................................ 13
Table 4: Test specification of adding organization unit ............................................................................................................. 14
Table 5: Test frames of adding organization unit, generated by the tool ................................................................................... 16
Table 6: Test specification of editing organization unit ............................................................................................................. 17
Table 7: Test frames of editing organization unit, generated by the tool ................................................................................... 18
Table 8: Test specification of deleting organization unit ............................................................................................................. 18
Table 9: Test frames of deleting organization unit, generated by the tool ................................................................................... 19
Table 10: Test specification of adding organization unit group ................................................................................................. 19
Table 11: Test frames of adding organization unit group, generated by the tool ............................................................. 20
Table 12: Test specification of Test specification of updating organization unit group ............................................................ 20
Table 13: Test frames of Test specification of adding organization unit group, generated by the tool ............................................. 21
Table 14: Test specification of deleting organization unit group ............................................................................................... 21
Table 15: Test frames of deleting organization unit group, generated by the tool ........................................................................ 22
Table 16: Test specification of adding group set ....................................................................................................................... 22
Table 17: Test frames of group set, generated by the tool ......................................................................................................... 23
Table 18: Test specification of editing group set ....................................................................................................................... 24
Table 19: Test frames of editing group set, generated by the tool ............................................................................................. 25
Table 20: Test specification of deleting group set ....................................................................................................................... 25
Table 21: Test frames of deleting group set, generated by the tool ............................................................................................... 26
Table 22: Test specification of hierarchy operators ................................................................................................................... 26
Table 23: Test frames of hierarchy operators, generated by the tool ........................................................................................... 27
Table 24: Test frames generated before and after changing the properties ................................................................................ 28
Table 25: Test frames generated before and after adding annotation Single to environment partition: name does not exist .... 28

List of abbreviation

HISP: Health Information System Program
DHIS2: District Health Information System
CPM: Category Partition Method
SUT: System Under Test
1 INTRODUCTION

1.1 Scope of project

This project aims to apply knowledge learned from the Software Verification and Validation given at University of Oslo, Spring 2010 to a practical case: the DHIS2 software, a tool for supporting data collection and processing targeted at health care sector. Testing is widely accepted as one of the most important activities to ensure quality of software (Mathur 2008). Given the increasing penetration of business management applications into all sectors, in which health care is one example, the need to have a proper and systematic approach to test those applications becomes very urgent.

1.2 Goals of project

DHIS2 has been used in many countries in Africa and Asia such as India, Tanzania, Serra Leon, and Vietnam. Hosted by University of Oslo under the umbrella of Health Information Program, DHIS2 is an OSS developed by a heterogeneous team with members coming from different domains: informatics and public health. At the beginning, testing was mainly done at the unit level by developers, using JUnit framework. Functional testing is executed in a very ad-hoc manner. There are not any documents related to test design. No one know how the functional test is done. Recently, as a response to the national implementation in India, the system was tested more rigorously. However, in order to be accepted as national system, DHIS2 must be sent to an assigned authority for software verification and validation to test. This step costs lots of money.

This story is to emphasize the urgent need to have a more systematic approach to test DHIS2. Hence, this study has two objectives: a) to develop a framework in order to build test sets systematically and thoroughly based on category partition technique, b) to develop a tool for automating test frames generation step, c) run the test set.

1.3 Structure of the report

The rest of this report is organized as follows. In the next section, the choices on techniques, methods, tools, and related work on black box testing are explained in section 2. Section 3 presents the system under test (SUT) while section 4 provide analysis of the results. Section 5 discusses the results and limitations of the study, and other open issues.

2 BACKGROUND

Category partition, one of black box methods, was selected to carry out the study. The following discussion will explain this choice.

2.1 Why black box?

Poon (2008) postulates that among those steps of software verification and validation, test case generation plays an important role in effecting the chance of discovering software failures. According to him, black box approach is the mainstream type of technique for test case generator. As based on information derived from system specification documents, black box testing can be done without having the knowledge on how the system was built, therefore source code is not needed (Ostrand and Balcer 1988, Poon 2008). Moreover, specification documents can help to derive and generate test cases even before any code are written. As the aim of this study is to do functional testing, black box approach appears to be an appropriate choice.
2.2 Why category partition?

So far, there are many techniques for black box testing that have been developed (Ostrand and Balcer 1988). For example, the choice relation framework, the classification-tree methodology, domain testing and orthogonal arrays (Poon, 2008) or condition table, equivalence partitioning, cause-effect graphs, revealing sub domain (Ostrand and Balcer, 1988). Apart from this, Mathur (2008) also discusses boundary analysis and logical functions as the other techniques for black box testing.

Poon (2008) argues that all the techniques are similar in term of the process in which tests case are generated: 1) identify categories and choices 2) combine valid choices 3) build test cases based on valid combination.

However, category partition method (CPM) seems to be a better technique as it is “a systematic, specification based method that uses partitioning to generate functional tests for complex software systems” (Ostrand and Balcer 1988, p.677). CPM is a combination of the two techniques: equivalence partitioning and boundary analysis (Mathur 2008). CPM has become a very popular methods, widely adopted, studied, and improved by vendors and researchers. Amla and Amman (1992) suggest a Z specification approach for CPM based on set theory and predicate calculus. Chen et al (2003) develop a framework called choice relation for CPM, proposing a theoretical technique for consistency check and automatically reducing the relations. Amman and Offutt (1994) construct a method for deriving test frames in CPM by defining a minimal coverage criterion and supplying a general procedure for specifying test cases that satisfy the criterion. Offutt and Irvine (1995) call for a reconsideration of traditional methods for object-oriented software, arguing that CPM can be effective to uncover faults in object-oriented software. Most recently, Lionel et al (2009) apply machine-learning approach to refine test specification and test suites in CPM.

Cause effect graph (CEF) is not considered as a good method in practice as it is “replacing one complex representation with another” (Ostrand and Balcer 1988, p.684) due to its complicated presentation and hard to verify correctness.

Classification tree (CTM) is also a frequently mentioned method which “is self-development partly using and improving ideas from the category-partition method” (Grochtmann et al, 1993, p2). Grochtmann et al (1993) discussed the application of CTM by an in-house developed tool called Classification Tree Editor. However, this tool is still internal use and its commercialization is in plan.

2.3 Why test frame generator?

Large body of research related to CPM can be grouped into two approaches as the following:

1. Try to revise, improve, or adapt it in a particular domain

2. Try to develop or explore a tool for automating several steps involved in CPM

---

1 Indeed, Ostrand and Balcer 1988 has been cited more than 100 times

2 CEG is used by a number of companies, one of which has built a tool for intuitively making the graph (Moyorodi, 2003) though it is a commercial product. http://benderrbt.com/bendersoftware.htm#over
According to my knowledge, there has not yet been a tool for transferring categories/choices into test frames though it is considered as an easy task. This tool is supposed to read Test Specification, i.e. categories/choices/constraints, from an input file (text, xml, or probably database), and then combine them to make a list of test frames given that those test frames satisfy all the constraints defined in the input file. Poon (2008) carries out an empirical case study in which three system specification are selected to be categorized/partitioned by 40 informatics students at both undergraduate and graduate level. The result of the study shows that there are many mistakes in categories and choices. The reason for this is easy to understand: system specification is written in natural language and selecting categories/choices/constraints is dependent heavily on experience of test engineers.

As building categories and choices is very prone to error, the need for a tool to generate test frames based on test specification becomes urgent.

3 DESIGN OF THE CASE STUDY

3.1 Description of system under test (SUT)

District Health Information System version 2 (DHIS2) is an open source software based on popular Java Enterprise frameworks such as Spring, Hibernate, and Struts2, aiming to strengthen health information system in developing countries. DHIS2 is a software for collection, processing, and analysis of health data. The core data model of DHIS2 includes major classes as follows:

- Organization Unit: administrative unit or facility such as a country, province, district, hospital, and clinic
- Data Element: element to capture health problems or health services such as number of children who have BCG, number of pregnant women, number of HIV cases.
- Period: an interval of time such as yearly, quarterly, or monthly.
- Data Value: number that keeps value of a data element for one organization unit within a particular period.

DHIS2 follows modular and layer design. It is organized in the three layers as follows:

- DHIS API: contains all data model classes and interface for services which are implemented in the service layer
- DHIS Service: implementation of the interfaces declared in the API layer
- DHIS Web: employs MVC (Model View Controller) design (Struts2). The web layer is structured into modules (there are currently almost 20 web modules), which communicates through a common module called web-portal.

DHIS2 has more than 50 modules consisting thousands of classes. In this assignment, I plan to test only one of them. That is the dhis-web-maintainance-organisationunit module.

3.2 The dhis-web-maintainance-organisationunit module

This web module provides functionalities to add, update, and delete organization units, group, and group set, and other hierarchy operators such as moving organization unit. Several functionalities of the module are demonstrated in the following screenshots:
The form to add new organization unit:

![Form to add new organization unit](image)

**Figure 2: Form to add new organization unit**

To add new unit group
3.3 The requirement specification of the module

The detailed specification of the module is provided as follows:
<table>
<thead>
<tr>
<th>Specification</th>
<th>Name of functionalities</th>
<th>Detailed description</th>
</tr>
</thead>
</table>
| **Organization unit** | **Add new organization** | The home page of the module show a list of units. User click Add new organization link, a form will be showed with the following fields and their constraints:  
- Name: not null AND no space (blank) before and after AND length of string name must be between 2 and 255 AND there is no unit with this name exists in the system  
- Short name: not null AND no space (blank) before and after AND length of string name must be between 2 and 25 AND there is no unit with this short name exists in the system  
- Code: any  
- Opening date: not null AND must be a valid date  
- Close date: can be null but if it is not null, it must contain no child unit  
- Comment: any  
- Polygon coordinator: any  
- Latitude: any  
- Longitude: any  
If all the above constraints are satisfied, the system will add this unit into the database and return to the list of unit page. If not, it will show a warning message telling users what conditions are violated.  
Before clicking the Add unit link, users shall select an unit on the left hand side hierarchy tree, this selected unit would become the parent unit of the to be added unit. If no unit is selected, the newly added unit will be the root unit. |
| **Edit an organization unit** | | In the unit listing page, each unit has different link for editing, deleting, and showing detailed information. Users click on an Edit link to edit the selected unit. The unit editing form will be showed, in line with all properties of the unit.  
Users update one or many of those properties and click update to save. However, updated properties must fulfill the |


### Delete an organization unit

In the unit listing page, select the Delete link, the system shall ask users whether they really want to delete the unit.

If users choose Yes, the unit is removed from the system and return to the unit-listing page. If not, the page remains the same.

Unit is removable only if it contains no child.

### Unit group

#### Add unit group

To add a new unit group, users need to give a name and select a list of organization units. The name must be at least two characters and less than 255 characters. Blank characters before and after the name must be removed when storing. The list of organization unit can be empty.

#### Edit unit group

Users have to fill a form which is similar to the add unit group functionality. In the edit form, users can remove one/many organization units from the list or add more one/many of them.

#### Delete unit group

The program must ask users for confirmation. Unit group can be removed only when it contains no organization unit.

### Group set (group of group)

#### Add a group set

To add a group set, users have to provide name, description, and a property to classify the group set as a compulsory or not. The length of name and description is between 2 and 255. Compulsory property has a combo box with two predefined value Yes and No.

If the compulsory property is Yes, there must be one organization unit group to be selected. Otherwise, no unit group is required.

#### Edit a group set

The requirement of input values is similar to adding a group set function.

#### Delete a group set

The program must ask users for confirmation. Unit group set can be deleted only when it contains no organization unit.
group

Hierarchy operators

| Move an organization unit | Users select an organization unit called orgA to be moved and after that select a parent organization unit for orgA. If no parent organization unit is selected, orgA will become a root. An organization unit can not be moved to itself. If orgA has children, they will also be moved with it. |

Table 1: Module specification

3.4 Building an automation tool for generating test frames based on test specification using CPM

3.4.1 Standardization of test specification language (TSL)

TSL was first proposed by Ostrand and Balcer (1988). According to this language, categories and choices are presented as a list of rows. Properties of choices are specified by the syntax [property X, Y] with X and Y as properties separated by a comma. There are two special annotations [error] and [single] which are used to reduce the number of possible combinations. The [error] property assigned to the choices that represent error condition while the [single] property assigned to the choices that are not to be combined with choices of other parameters (Mathur 2008). Though the TSL proposed by Ostrand and Balcer (1988) is simple and easy to understand, it has some limitations. First, the characters [ and ] used for describing properties of choices take testers longer time to type. Second, it is written as a text file, therefore, difficult for a program to process.

3.4.2 A revised TSL

To make it more effective, I propose a modified version of TSL in order to overcome the challenges in the original TSL. The modified TSL employs a column-oriented approach that places category, choice, property, and condition into different columns. With this approach, testers can save certain amount of time in typing the special characters [, ], but still keep the readability of the language. The modified TSL is described in the following table:

<table>
<thead>
<tr>
<th>STT</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Category, partition, property, and constraints are placed in four different columns</td>
</tr>
<tr>
<td>2</td>
<td>Remove the open and close character: [ and ]</td>
</tr>
<tr>
<td>3</td>
<td>Remove the keyword: if</td>
</tr>
</tbody>
</table>
3.4.3 Using Excel file as input/output

There are different alternatives for input formats such as hierarchy structure xml, comma separated value (csv) file, relational database structure, and excel. I decided to select Excel because of several reasons.

First, it is one of the most common used programs including browser, and word processor. To make it possible for testers who do not have deep technical understanding to use this category partition method, Excel appears to be a good choice.

Second, time to produce test specification should be minimized. Building categories, participation, and their constraints is not always a straightforward process. Easy re-factoring, i.e. changing or updating partitions and constraints, can help to increase chance to have good test sets and allow testers to save time in unnecessary steps.

Third, Excel allows a systematic way of storing test specification and test set, easy to be retrieved and transferred.

3.4.4 Building algorithm

- Class design:

The tool consists of the following classes:

  a. Category
  b. Choice
  c. Property
  d. CPGenerator

Each category has a list of choices and each choice has a list of properties and conditions. The CPGenerator is the main class that read the Excel file, build the data structure from the file, and create the test frames.

- Algorithm: the CPGenerator class has a main method called buildTestFrames and other supporting methods. The algorithm for buildTestFrames method can be described in pseudo code as follows:

  a. Reading the input Excel file
  b. Building a data structure of the test specification from the Excel file
  c. Create test frame with error annotation
  d. Create test frame with single annotation
  e. Create test frame remaining

Test frames created in step c, d, and e are inserted into a new Excel file.

3.5 Initial results

In this section, I applied the tool developed in the previous section to test the SUT described in the case study. The following steps have been done:
- Build the categories, choices, and constraints using the revised TSL (Excel file as input)
- Apply the tool for the case to build the test set (Excel file as output)
- Run the test set manually

The test specification, the test frame generated by the tool, and the results of test cases are presented as follows:

3.5.1 Organization unit functionalities

Adding:

Test specification

<table>
<thead>
<tr>
<th>Categories</th>
<th>Partitions</th>
<th>Properties</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>openingdate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>null</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>valid</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>notvalid</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>closingdate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>null</td>
<td>single</td>
<td></td>
</tr>
<tr>
<td></td>
<td>valid and &gt;= openingdate</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>valid and &lt; openingdate</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>notvalid</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>selectedorg</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>null</td>
<td>single</td>
<td></td>
</tr>
<tr>
<td></td>
<td>notnull</td>
<td></td>
<td></td>
</tr>
<tr>
<td>environment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>name not exist</td>
<td>nameok</td>
<td></td>
</tr>
<tr>
<td></td>
<td>shortcode not exist</td>
<td>shortcodeok</td>
<td></td>
</tr>
<tr>
<td></td>
<td>name exist</td>
<td>nameok</td>
<td></td>
</tr>
<tr>
<td></td>
<td>shortcode exist</td>
<td>shortcodeok</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Test specification of adding organization unit

Test frames

<table>
<thead>
<tr>
<th>No</th>
<th>Test frame ID</th>
<th>name</th>
<th>shortcode</th>
<th>openingdate</th>
<th>selectedorg</th>
<th>environment</th>
<th>Expected output</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>length=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Error message on invalid input</td>
<td>P</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>length=1</td>
<td></td>
<td></td>
<td>Error message on invalid input</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>length&gt;255</td>
<td></td>
<td></td>
<td>Error message on invalid input</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>length=0</td>
<td></td>
<td></td>
<td>Error message on invalid input</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>length&gt;25</td>
<td></td>
<td></td>
<td>Error message on invalid input</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>null</td>
<td></td>
<td></td>
<td>Error message on invalid input</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>notvalid</td>
<td></td>
<td></td>
<td>Error message on invalid input</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3.2.2.1.1.1. length=2-255</td>
<td>valid</td>
<td>null</td>
<td>Add successfully a root orgunit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3.2.2.2.1.1. length=2-255</td>
<td>valid</td>
<td>notnull</td>
<td>Add successfully a non-root orgunit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>3.2.2.2.2.1.1. length=2-255</td>
<td>valid</td>
<td>notnull</td>
<td>Add successfully a non-root</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 4: Test frames of adding organization unit, generated by the tool

**Editing: the selected organization input is not available**

Test specification

<table>
<thead>
<tr>
<th>Categories</th>
<th>Partitions</th>
<th>Properties</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>length=0</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>length=1</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>length=2-255</td>
<td>nameok</td>
<td></td>
</tr>
<tr>
<td></td>
<td>length&gt;255</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>shortname</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>length=0</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>length=2-25</td>
<td>shortnameok</td>
<td></td>
</tr>
<tr>
<td></td>
<td>length&gt;25</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>openingdate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>null</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>valid</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>notvalid</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>closingdate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>null</td>
<td>single</td>
<td></td>
</tr>
<tr>
<td></td>
<td>valid and &gt;= openingdate</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>valid and &lt; openingdate</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>notvalid</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>environment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>name not exist</td>
<td>nameok</td>
<td></td>
</tr>
<tr>
<td>Test frame</td>
<td>name</td>
<td>shortname</td>
<td>openingdate</td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>length=0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>length=1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>length&gt;255</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>length=0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>length&gt;25</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>notvalid</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3.2.2.1.1.1.1</td>
<td>length=2-255</td>
<td>length=2-25</td>
</tr>
</tbody>
</table>

Table 5: Test specification of editing organization unit
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td></td>
<td>valid and &lt; openingdate</td>
<td>successfully orgunit</td>
<td>Error message on invalid input</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>not valid</td>
<td>Error message on invalid input</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>3.2.2.1.1.1. length=2-255 length=2-25 valid</td>
<td>valid and &gt;= openingdate name not exist</td>
<td>Update successfully orgunit</td>
<td>P</td>
</tr>
<tr>
<td>12</td>
<td>3.2.2.2.2.1.1. length=2-255 length=2-25 valid</td>
<td>valid and &gt;= openingdate shortname not exist</td>
<td>Update successfully orgunit</td>
<td>P</td>
</tr>
<tr>
<td>13</td>
<td>3.2.2.3.1.1. length=2-255 length=2-25 valid</td>
<td>valid and &gt;= openingdate name exist</td>
<td>Error message on existing name</td>
<td>P</td>
</tr>
<tr>
<td>14</td>
<td>3.2.2.4.1.1. length=2-255 length=2-25 valid</td>
<td>valid and &gt;= openingdate shortname exist</td>
<td>Error message on existing name</td>
<td>P</td>
</tr>
</tbody>
</table>

Table 6: Test frames of editing organization unit, generated by the tool

Deleting

Test specification

<table>
<thead>
<tr>
<th>Categories</th>
<th>Partitions</th>
<th>Properties</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>environment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>orgunit contains a child</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>orgunit contains no child</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Test specification of deleting organization unit

Test frames
### Test frames of deleting organization unit, generated by the tool

<table>
<thead>
<tr>
<th>No</th>
<th>Test frame ID</th>
<th>environment</th>
<th>Expected output</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.1.1.1.1.1.1.</td>
<td>orgunit contains a child</td>
<td>Fail to delete</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2.1.1.1.1.1.1.</td>
<td>orgunit contains no child</td>
<td>Succeed to delete</td>
<td></td>
</tr>
</tbody>
</table>

**Table 8: Test frames of deleting organization unit, generated by the tool**

### 3.5.2 Organization unit group

#### Adding

**Test specification**

<table>
<thead>
<tr>
<th>Categories</th>
<th>Partitions</th>
<th>Properties</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>length=0</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>length=1</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>length=2-255</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>length&gt;255</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>list of orgunit</td>
<td>0 orgunit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 orgunit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 orgunit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>environment</td>
<td>name does not exist</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>name exists</td>
<td></td>
<td>single</td>
</tr>
</tbody>
</table>

**Table 9: Test specification of adding organization unit group**

**Test frames**

<table>
<thead>
<tr>
<th>No</th>
<th>Test frame ID</th>
<th>name</th>
<th>list of orgunit</th>
<th>environment</th>
<th>Expected output</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>length=0</td>
<td></td>
<td></td>
<td>Error message on invalid input</td>
<td>P</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>length=1</td>
<td></td>
<td></td>
<td>Error message on invalid input</td>
<td>P</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>length&gt;255</td>
<td></td>
<td></td>
<td>Error message on invalid input</td>
<td>F</td>
</tr>
</tbody>
</table>
Table 10: Test frames of Test specification of adding organization unit group, generated by the tool

### Editing

Test specification

<table>
<thead>
<tr>
<th>Categories</th>
<th>Partitions</th>
<th>Properties</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>length=0</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>length=1</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>length=2-255</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>length&gt;255</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>list of orgunit</td>
<td>0 orgunit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 orgunit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 orgunit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>environment</td>
<td>name exists</td>
<td>single</td>
<td></td>
</tr>
<tr>
<td></td>
<td>name does not exist</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 11: Test specification of Test specification of updating organization unit group

Test frames

<table>
<thead>
<tr>
<th>No</th>
<th>Test frame ID</th>
<th>name</th>
<th>list of orgunit</th>
<th>environment</th>
<th>Expected output</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>length=0</td>
<td></td>
<td>Error message on invalid input</td>
<td>P</td>
</tr>
<tr>
<td>2</td>
<td>length=1</td>
<td>Error message on invalid input</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>length&gt;255</td>
<td>Error message on invalid input</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3.1.1.1.1.1.1. length=2-255 0 orgunit name exists</td>
<td>Error message on existing name</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3.1.2.1.1.1.1. length=2-255 0 orgunit name does not exist</td>
<td>Update successfully group with 0 orgunit</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3.2.2.1.1.1.1. length=2-255 1 orgunit name does not exist</td>
<td>Update successfully group with 1 orgunit</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3.3.2.1.1.1.1. length=2-255 2 orgunit name does not exist</td>
<td>Update successfully group with 2 orgunit</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 12: Test frames of Test specification of adding organization unit group, generated by the tool

**Deleting**

Test specification

<table>
<thead>
<tr>
<th>Categories</th>
<th>Partitions</th>
<th>Properties</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>environment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>group has 1 orgunit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>group has 0 orgunit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 13: Test specification of deleting organization unit group

Test frame

<table>
<thead>
<tr>
<th>No</th>
<th>Test frame ID</th>
<th>environment</th>
<th>Expected output</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.1.1.1.1.1.1.</td>
<td>group has 1 orgunit</td>
<td>Fail to delete</td>
<td>P</td>
</tr>
</tbody>
</table>
Table 14: Test frames of deleting organization unit group, generated by the tool

<table>
<thead>
<tr>
<th>Group set</th>
<th>group has 0 orgunit</th>
<th>Succeed to delete</th>
<th>P</th>
</tr>
</thead>
</table>

### 3.5.3 Group set

#### Adding

Test specification

<table>
<thead>
<tr>
<th>Categories</th>
<th>Partitions</th>
<th>Properties</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>length=0</td>
<td></td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>length=1</td>
<td></td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>length=2-255</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>length&gt;255</td>
<td></td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>length=0</td>
<td></td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>length=1</td>
<td></td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>length=2-255</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>length&gt;255</td>
<td></td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>compulsory</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>list of group</th>
<th>environment</th>
<th></th>
<th>Expected output</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 group</td>
<td></td>
<td></td>
<td>Error message on invalid input</td>
<td>P</td>
</tr>
<tr>
<td>2 groups</td>
<td></td>
<td>single</td>
<td>Error message on invalid input</td>
<td>P</td>
</tr>
<tr>
<td>0 group</td>
<td></td>
<td></td>
<td>Error message on invalid input</td>
<td>P</td>
</tr>
</tbody>
</table>

Table 15: Test specification of adding group set

Test frame

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
<th>compulsory</th>
<th>list of group</th>
<th>environment</th>
<th>Expected output</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>length=0</td>
<td></td>
<td></td>
<td></td>
<td>Error message on invalid input</td>
<td>P</td>
</tr>
<tr>
<td>2</td>
<td>length=1</td>
<td></td>
<td></td>
<td></td>
<td>Error message on invalid input</td>
<td>P</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>----------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>length&gt;255</td>
<td></td>
<td></td>
<td>Error message on invalid input P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>length=0</td>
<td></td>
<td></td>
<td>Error message on invalid input P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>length=1</td>
<td></td>
<td></td>
<td>Error message on invalid input P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>length&gt;255</td>
<td></td>
<td></td>
<td>Error message on invalid input F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3.3.1.2.1.1.</td>
<td>length=2-255</td>
<td>length=2-255</td>
<td>Yes 2 groups, group set name does not exist Add successfully group set with 2 groups P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3.3.1.1.1.1.</td>
<td>length=2-255</td>
<td>length=2-255</td>
<td>Yes 1 group, group set name exist Error message on existing name P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3.3.1.1.2.1.1.</td>
<td>length=2-255</td>
<td>length=2-255</td>
<td>Yes 1 group, group set name does not exist Error message when compulsory is Yes but no group is selected P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>3.3.1.3.1.1.</td>
<td>length=2-255</td>
<td>length=2-255</td>
<td>Yes 0 group, group set name does not exist Add successfully group set with 1 groups P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>3.3.2.1.1.1.</td>
<td>length=2-255</td>
<td>length=2-255</td>
<td>No 1 group, group set name does not exist Add successfully group set with 1 groups P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>3.3.2.3.1.1.</td>
<td>length=2-255</td>
<td>length=2-255</td>
<td>No 0 group, group set name does not exist Add successfully group set with 0 groups P</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 16: Test frames of group set, generated by the tool**

**Editing**

Test specification

<table>
<thead>
<tr>
<th>Categories</th>
<th>Partitions</th>
<th>Properties</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td></td>
<td>length=0</td>
<td>error</td>
</tr>
<tr>
<td></td>
<td>length=1</td>
<td>error</td>
<td>length=2-255</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>length=0</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>length=1</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>length&gt;255</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>length=0</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>length=1</td>
<td>error</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>length&gt;255</td>
<td>error</td>
<td></td>
</tr>
</tbody>
</table>

Table 17: Test specification of editing group set

Test frame
### Table 18: Test frames of editing group set, generated by the tool

<table>
<thead>
<tr>
<th>Test frame</th>
<th>Categories</th>
<th>Partitions</th>
<th>Properties</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>3.3.1.</td>
<td>2.2.1.</td>
<td>length=2-255</td>
<td>length=2-255</td>
</tr>
<tr>
<td>8</td>
<td>3.3.1.</td>
<td>1.1.1.</td>
<td>length=2-255</td>
<td>length=2-255</td>
</tr>
<tr>
<td>9</td>
<td>3.3.1.</td>
<td>1.2.1.</td>
<td>length=2-255</td>
<td>length=2-255</td>
</tr>
<tr>
<td>10</td>
<td>3.3.1.</td>
<td>3.2.1.</td>
<td>length=2-255</td>
<td>length=2-255</td>
</tr>
<tr>
<td>11</td>
<td>3.3.2.</td>
<td>1.2.1.</td>
<td>length=2-255</td>
<td>length=2-255</td>
</tr>
<tr>
<td>12</td>
<td>3.3.2.</td>
<td>3.2.1.</td>
<td>length=2-255</td>
<td>length=2-255</td>
</tr>
</tbody>
</table>

### Table 19: Test specification of deleting group set

**Deleting**

Test specification

<table>
<thead>
<tr>
<th>Environment</th>
<th>Group Set Contains a Child</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>group set contains a child</td>
<td></td>
</tr>
<tr>
<td></td>
<td>group set contains no child</td>
<td></td>
</tr>
</tbody>
</table>

Test frame

<table>
<thead>
<tr>
<th>Test frame</th>
<th>Environment</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.1.1.1.1.1</td>
<td>Fail to delete</td>
</tr>
</tbody>
</table>
2.1.1.1.1.1.
group set
contains no child
Succeed to delete

Table 20: Test frames of deleting group set, generated by the tool

3.5.4 Hierarchy operators

Test specification

<table>
<thead>
<tr>
<th>Categories</th>
<th>Partitions</th>
<th>Properties</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>orgunit to be moved</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>non-root orgunit</td>
<td></td>
<td></td>
<td>nonroot</td>
</tr>
<tr>
<td>root orgunit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>target orgunit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>same as the moving orgunit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>child of the moving orgunit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parent of the moving orgunit</td>
<td></td>
<td></td>
<td>nonroot</td>
</tr>
<tr>
<td>brother of the moving orgunit</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 21: Test specification of hierarchy operators

Test frame

<table>
<thead>
<tr>
<th>Test frame</th>
<th>orgunit to be moved</th>
<th>target orgunit</th>
<th>Expected output</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.1.1.1.1.1.1.1.</td>
<td>non-root orgunit</td>
<td>same as the moving orgunit</td>
<td>Error message about one group can not be moved to itself</td>
</tr>
<tr>
<td>2</td>
<td>1.2.1.1.1.1.1.</td>
<td>non-root orgunit</td>
<td>child of the moving orgunit</td>
<td>Error message about one group can not be moved to its children</td>
</tr>
<tr>
<td>3</td>
<td>1.3.1.1.1.1.1.</td>
<td>non-root orgunit</td>
<td>parent of the moving orgunit</td>
<td>Successful but unchanged</td>
</tr>
<tr>
<td>4</td>
<td>1.4.1.1.1.1.1.</td>
<td>non-root orgunit</td>
<td>brother of the moving orgunit</td>
<td>Successful move</td>
</tr>
<tr>
<td>5</td>
<td>2.1.1.1.1.1.1.</td>
<td>root orgunit</td>
<td>same as the moving orgunit</td>
<td>Error message about one group can not be moved to itself</td>
</tr>
<tr>
<td>6</td>
<td>2.2.1.1.1.1.1.</td>
<td>root orgunit</td>
<td>child of the moving orgunit</td>
<td>Error message about one group can not be moved to its children</td>
</tr>
</tbody>
</table>
2.4.1.1.1.1. root orgunit brother of the moving orgunit Successful move P

Table 22: Test frames of hierarchy operators, generated by the tool

4 ANALYSIS OF THE RESULTS

4.1 Analyzing the results

4.1.1 Effectiveness of CPM:

By using the tool, a test set was created comprising 87 test cases for four sub-functionalities: organization unit, group, group set, and hierarchy operator. This number of test cases is feasible for testers to test in practice given the time constraint. It is also very much smaller compared to the number of possible test cases when not using CPM. The equivalence partition method would result in a huge number of test cases. For example, in the adding organization unit functionality alone, there are 5 categories containing 4, 3, 3, 2, 4 partitions respectively, the number of test case for this situation is 4x3x3x2x4 = 288 test cases. For all the functionalities of the organization unit management module, number of test cases can end up at thousands. This is not feasible to test in practice.

4.1.2 Effectiveness of the tool:

During the case study, the tool has helped me to rebuild the test frames quickly whenever I discovered a mistake in defining categories and choices.

For example, the following table shows how the tool can help to rebuild the test frame table if testers make a mistake in writing test specification. This table is related to the hierarchy operator functionality.

In the specification Before, I made mistake when forgetting to define a constraint for orgunit to be moved and the target orgunit. The consequence was a test frame which combined root orgunit and its parent was built. However, this combination was invalid and could not possible in practice. By creating a constraint, as showed in the specification After, the test frames did not contain that ill-logical combination.

<table>
<thead>
<tr>
<th>Specification</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Categories</td>
<td>Partitions</td>
<td>Properties</td>
</tr>
<tr>
<td>orgunit to be moved</td>
<td>non-root orgunit</td>
<td></td>
</tr>
<tr>
<td>root orgunit</td>
<td>root orgunit</td>
<td></td>
</tr>
<tr>
<td>target orgunit</td>
<td>same as the moving orgunit</td>
<td></td>
</tr>
<tr>
<td>child of the moving orgunit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>parent of the moving orgunit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>brother of the moving orgunit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>orgunit to be moved</td>
<td>non-root orgunit</td>
</tr>
<tr>
<td></td>
<td>target orgunit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>same as the moving orgunit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>child of the moving orgunit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>parent of the moving orgunit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>brother of the moving orgunit</td>
<td></td>
</tr>
</tbody>
</table>
Table 23: Test frames generated before and after changing the properties

Another example of this kind of mistakes was in the updating group functionality which is summarized in the following table:

<table>
<thead>
<tr>
<th>No</th>
<th>orgunit to be moved</th>
<th>target orgunit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>non-root orgunit</td>
<td>same as the moving orgunit</td>
</tr>
<tr>
<td>2</td>
<td>non-root orgunit</td>
<td>child of the moving orgunit</td>
</tr>
<tr>
<td>3</td>
<td>non-root orgunit</td>
<td>parent of the moving orgunit</td>
</tr>
<tr>
<td>4</td>
<td>non-root orgunit</td>
<td>brother of the moving orgunit</td>
</tr>
<tr>
<td>5</td>
<td>root orgunit</td>
<td>same as the moving orgunit</td>
</tr>
<tr>
<td>6</td>
<td>root orgunit</td>
<td>child of the moving orgunit</td>
</tr>
<tr>
<td>7</td>
<td>root orgunit</td>
<td>parent of the moving orgunit</td>
</tr>
<tr>
<td>8</td>
<td>root orgunit</td>
<td>brother of the moving orgunit</td>
</tr>
</tbody>
</table>

Table 24: Test frames generated before and after adding annotation Single to environment partition: name does not exist

<table>
<thead>
<tr>
<th>No</th>
<th>name</th>
<th>list of orgunit</th>
<th>environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>length=0</td>
<td>0 orgunit</td>
<td>name exists</td>
</tr>
<tr>
<td>2</td>
<td>length=1</td>
<td>1 orgunit</td>
<td>name does not exist</td>
</tr>
<tr>
<td>3</td>
<td>length&gt;255</td>
<td>2 orgunit</td>
<td>name does not exist</td>
</tr>
<tr>
<td>4</td>
<td>length=2-255</td>
<td>0 orgunit</td>
<td>name exists</td>
</tr>
<tr>
<td>5</td>
<td>length=2-255</td>
<td>1 orgunit</td>
<td>name exists</td>
</tr>
<tr>
<td>6</td>
<td>length=2-255</td>
<td>2 orgunit</td>
<td>name does not exist</td>
</tr>
<tr>
<td>7</td>
<td>length=2-255</td>
<td>1 orgunit</td>
<td>name does not exist</td>
</tr>
</tbody>
</table>
The table shows that the **Before** Specification led to three redundant test frames which are marked as red color. When the group name exists in the system, test cases combing that existing group name with three different options of list size are not useful at all because the system behaves the same: checking and showing error message on the existing name. By using the tool, new test frames table was re-generated quickly by making a change in the test specification: adding [single] to the choice “name exists”.

4.1.3 **Results of running the test set:**

Manual running the test set built the previous section produced the results with six failed test cases. All of them are related to the length of input value. 

The defect rate = \( \frac{6}{87} \times 100 = 6.98\% \)

This result can derive several notices:

- The organization unit management module is an advanced module and mostly used by super users. In all the countries, this module was used exclusively by the HISP team. The organization unit hierarchy for each country was given as “factory setting”. If the module had been used by the end users, probably these bugs could have been reported.

- This is a very simple module but there are bugs on it. Applying CPM in other complicated modules could reveal more bugs.

- Applying method like CPM can increase the confidence in order to release the product. It also helps to estimate time for testing, hence, making better project planning.

4.1.4 **The relations between graphical user interface (GUI) design and CPM:**

Mathur (2008) have several discussions on the relations between GUI and equivalence class. He argues that if the GUI can prevent invalid input, there is no need for equivalence class on such invalid values (p.110, p.117). It means that testers do not need to build such test cases. I think that argument is also applied in the case of CPM. For example, in the Group Set functionality, there is a property called Compulsory. This property is implemented as a combo box with two options Yes or No. With this design, there is no chance for users to input invalid values such as numbers and other strings. Therefore, test design must take into account GUI design (Mathur 2008) in order to have a small and feasible test sets. On the other hand, the GUI design should also be based on test design (Mathur 2008).

It is also worth to notice that in the adding organization unit functionality, the property Open Date and Closing Date are slightly different from the case described above. These properties accept only valid date. However, they allow users to either select a date from a date picker or type in a text box. The reason for this is because typing a date is faster compared to selecting from the date picker. As a result, testers have to build test cases based on the both of the use cases: selecting date from date picker and free text typing.
4.2 Summary of important results

- The case study shows that CPM is a very effective method especially for web based and enterprise application such as DHIS2. When several techniques of white box testing seem difficult to apply, black box testing technique becomes an obvious choice.

- A tool has been developed to automate the test frame building process from the test specification. This tool allows testers to define categories, choices, and constraints quickly with simple Excel file as input.

- A module of DHIS2 software - organization unit module - has been used as an empirical case to experiment the effectiveness of the tool. As presented in the case study section, the tool appears to be productive in helping testers to save time in building test sets and increase quality of test cases by offering the refactoring functionalities, i.e. testers can re-build test frames when changing the test specification.

5 LESSONS LEARNED AND OPEN ISSUES

5.1 Lesson learned

Through this project, I have learned many things which will be useful for my work:

- The important role of testing in producing high quality software
- Assess the adequacy of a test set based on coverage criteria. Know how to improve the quality of test set.
- Know how to apply systematic testing to a particular system
- Partially address the problem of quality in DHIS2 software. The result of this project can serve as a departure to apply testing for the rest of modules in DHIS2.

5.2 Practical and technical difficulties

While doing this assignment, I encountered several difficulties. The first challenge I find hard to overcome is how to select a good project. Testing is a very broad topic with many different approaches, techniques, and categories. Therefore selecting appropriate methods requires experience but also contains risks. There is a big chance for selecting wrong approaches. It is hard to find a neither too big nor too small system in order to test. Therefore, I have to select a business web application.

Second, tools for testing are plenty but not-all-of-them are good. Many OSS testing projects are outdated, creating difficulties on acquiring supports and being compatible of newer versions of other software, i.e. JDK 1.6. Working through all the tools require enormous amount of time. Moreover, there are lacking lots of tools for automatically generating test cases based on test specification. For example, there is no tool for common used methods like Category Partition Method and Cause Effect Graph Method. Indeed, there is tool called BenderRBT for cause effect graph but it is not OSS (http://benderrbt.com/)

Available mutation testing tools such as MuClipse, Jester etc do not support Spring Dependency Injection, i.e. which class to be called is declared in a applicationContext.xml file. Mutation tools while creating new mutant class do...
not make the change in the applicationContext.xml file accordingly making the mutant class unable to run. This is practically
difficult for those who select white box testing on Spring based web application.

Third, analyzing the results of the case study seems very difficult task. I was very confused about what contribution this
project can make. Is it to find as many bug of the SUT as possible? Or should it aim to prove that a certain method is better
than another by using empirical data from the case study? Or a combination of all?

5.3 Discussion, limitation, and future research

The tool can help test engineers build the test set quickly; however, there is a large space open for future research.

- The tool needs to be extended in order to address the issue of limitation in the number of inputs. Currently only
  maximum seven inputs are allowed.

- The process to automatically generate test script from test frames as described in Step 8 in Ostrand and Balcer -
  1988 is not possible in this case study. I could not manage do find a systematic way to run the test automatically.
  Though I could manually record test executions by using Selenium\(^4\), save them in Java code, and run them later.

- More research needs to be done to compare the productivity and the quality of the test sets between two groups of
testers: using the tool and not using the tool.

Due to the time constraints, several ideas have not been implemented in this project. If I could have had more time, I will try
to do the following things:

- Compare Category Partition with Cause Effect Graph (CEG) to see which method is better in detecting bugs. I also
  plan to make a tool to build the cause-effect graph using language like XML. I believe CEG can enable automating
test case generation because each Cause has only two values: true or false. By assigning each case (true or false) a
  particular value, it is possible to develop a tool to read these values to build test cases.

- Run the test cases developed by CPM against the source code to measure the coverage of code executed by these
test cases. This combination of black box and white box techniques could create a better understanding the
(possible) relationship between the two approaches. Therefore, in the case of available source code, testers can know
the adequacy of their test sets.

\(^4\) [http://seleniumhq.org/](http://seleniumhq.org/)
6 REFERENCES

Book:


Paper

T. Y. Chen and Pak-Lok Poon, Experience With Teaching Black-Box Testing in a Computer Science/Software Engineering Curriculum, IEEE TRANSACTIONS ON EDUCATION, VOL. 47, NO. 1, FEBRUARY 2004

Xu Baowen* ** Nie Changhai* Shi Qunfeng* Lu Hong*, AN ALGORITHM FOR AUTOMATICALLY GENERATING BLACK-BOX TEST CASES 1, Vol.20 No.1 JOURNAL OF ELECTRONICS Jan. 2003


Beizer, B.; Software Testing Techniques, Van Nostrand Reinhold, USA, 1990


7 APPENDICES

7.1 The source code of the tool developed in the project

Category.java

```java
package org.hisp.dhis.oum.functional.automation;
import java.util.Collection;

public class Category {
    private String name;
    private int position;
    private int cellPosition;
    private Collection<Choice> partitions;
    private int start;
    private int end;

    public Category() {
        super();
    }

    public Category(String name) {
        super();
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getPosition() {
        return position;
    }

    public void setPosition(int position) {
        this.position = position;
    }

    public int getCellPosition() {
        return cellPosition;
    }

    public void setCellPosition(int cellPosition) {
        this.cellPosition = cellPosition;
    }

    public Collection<Choice> getPartitions() {
        return partitions;
    }

    public void setPartitions(Collection<Choice> partitions) {
        this.partitions = partitions;
    }

    public int getStart() {
        return start;
    }

    public void setStart(int start) {
        this.start = start;
    }

    public int getEnd() {
        return end;
    }

    public void setEnd(int end) {
        this.end = end;
    }
}
```

Choice.java

```java
package org.hisp.dhis.oum.functional.automation;
```
import java.beans.PropertyVetoException;
import java.util.Collection;

public class Choice {
    private String name;
    private int position;
    private int cellPosition;
    private Category category;
    private Property property;
    private String sProperty;
    private String sCondition;
    private Collection<String> conditions;
    private Collection<Property> properties;

    public Collection<Property> getProperties() {
        return properties;
    }

    public void setProperties(Collection<Property> properties) {
        this.properties = properties;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getPosition() {
        return position;
    }

    public void setPosition(int position) {
        this.position = position;
    }

    public int getCellPosition() {
        return cellPosition;
    }

    public void setCellPosition(int cellPosition) {
        this.cellPosition = cellPosition;
    }

    public Category getCategory() {
        return category;
    }

    public void setCategory(Category category) {
        this.category = category;
    }

    public Property getProperty() {
        return property;
    }

    public void setProperty(Property property) {
        this.property = property;
    }

    public Collection<String> getConditions() {
        return conditions;
    }

    public void setConditions(Collection<String> conditions) {
        this.conditions = conditions;
    }

    public String getSProperty() {
        return sProperty;
    }

    public void setSProperty(String property) {
        sProperty = property;
    }

    public String getSCondition() {
        return sCondition;
    }

    public void setSCondition(String condition) {
        sCondition = condition;
    }
}
public void setSCondition(String condition) {
    sCondition = condition;
}
}

Property.java
package org.hisp.dhis.oum.functional.automation;
public class Property {
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Property(String name) {
        super();
        this.name = name;
    }
}

CPGenerator.java
package org.hisp.dhis.oum.functional.automation;
import java.io.File;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Date;
import java.util.List;
import jxl.Cell;
import jxl.Sheet;
import jxl.Workbook;
import jxl.write.Label;
import jxl.write.WritableCellFormat;
import jxl.write.WritableFont;
import jxl.write.WritableSheet;
import jxl.write.WritableWorkbook;
public class CPGenerator {
    private static String ERROR = "error";
    private static String SINGLE = "single";
    public int buildTestFrames(String dir, String inputFile, int sheet) throws Throwable {
        // TODO: Implementation
    }
    // Other methods and classes
}
```java
int k=0;
for(Cell cell: cells)
{
    if(cell.getContents().trim().length()>0)
    {
        Category category = new Category(cell.getContents());
        category.setPosition(cell.getRow());
        categories.add(category);
    }
    k++;
}
Cell[] cellPartitions = rsheet.getColumn(1);
Cell[] cellProperties = rsheet.getColumn(2);
Cell[] cellConditions = rsheet.getColumn(3);
int j=0;
for(int i=0;i<categories.size();i++)
{
    Category cat = categories.get(i);
    List<Choice> partitions = new ArrayList<Choice>();
    j= categories.get(i).getPosition()+1;
    int parid = 1;
    while(cellPartitions[j].getContents().length()>0)
    {
        Choice partition = new Choice();
        partition.setPosition(parid);
        partition.setName(cellPartitions[j].getContents().trim());
        partition.setSProperty(cellProperties[j].getContents().trim());
        partition.setSCondition(cellConditions[j].getContents().trim());
        partition.setCategory(cat);
        //deprecated
        String propertyName = cellProperties[j].getContents().trim();
        if(propertyName.trim()!=null)
        {
            partition.setProperty(new Property(propertyName));
        }
        //inserting list of properties
        String property = cellProperties[j].getContents().trim();
        List<Property> properties = new ArrayList<Property>();
        if(property.trim()!=null)
        {
            String[] proarray = property.split(",");
            for(String pro: proarray)
            {
                properties.add(new Property(pro.trim()));
            }
            partition.setProperties(properties);
        }
        //inserting list of conditions
        String condition = cellConditions[j].getContents();
        List<String> conditions = new ArrayList<String>();
        if(condition.trim()!=null)
        {
            String[] conarray = condition.split(",");
            for(String con: conarray)
            {
                conditions.add(con.trim());
            }
            partition.setConditions(conditions);
        }
        partitions.add(partition);
        parid++;
        j++;
    }
    categories.get(i).setPartitions(partitions);
    //break;
}
```
```java
int collumn = 2;
Label l = null;
for (Category category : categories) {
    // first setup the title
    l = new Label(collumn, 0, category.getName() , cf);
    s.addCell(l);
    collumn++;
}

// write expected output and pass/fail
l = new Label(collumn++, 0, "Expected output" , cf);
int testCaseNo = 1;
String testFrame = "\n";
for (Category category : categories) {
    for (Choice partition : category.getPartitions()) {
        if (partition.getName().trim().length() > 0 && partition.getProperty().getName().trim().equals("ERROR")) {
            l = new Label(0, testCaseNo, String.valueOf(testCaseNo) , cf);
            s.addCell(l);
            l = new Label(collumn, row, partition.getName() , cf);
            s.addCell(l);
            row++;
            testCaseNo++;
        }
        if (partition.getName().trim().length() > 0 && partition.getProperty().getName().trim().equals("SINGLE")) {
            l = new Label(0, testCaseNo, String.valueOf(testCaseNo) , cf);
            s.addCell(l);
            // for the rest of input
            int subcolumn = 2;
            for (Category cat : categories) {
                Choice par = getValidPartitionFromCategory(cat);
                if (par.getCategory().getName().equals(cat.getName())) {
                    l = new Label(collumn, row, par.getName() , cf);
                    s.addCell(l);
                    testFrame += par.getPosition() + ".";
                } else {
                    l = new Label(subcolumn, row, par.getName() , cf);
                    s.addCell(l);
                    testFrame += par.getPosition() + ".";
                    subcolumn++;
                }
            }
            // add test frame
            l = new Label(1, row, testFrame , cf);
            s.addCell(l);
            // add the single itself
            row++;
            testCaseNo++;
        }
    }
    collumn++;
} // for category
```
int count = 1;

List<List<Choice>> combinations = new ArrayList<List<Choice>>();

for (Choice par0: getCategoryById(categories, 0).getPartitions())
{
    for (Choice par1: getCategoryById(categories, 1).getPartitions())
    {
        for (Choice par2: getCategoryById(categories, 2).getPartitions())
        {
            for (Choice par3: getCategoryById(categories, 3).getPartitions())
            {
                for (Choice par4: getCategoryById(categories, 4).getPartitions())
                {
                    for (Choice par5: getCategoryById(categories, 5).getPartitions())
                    {
                        for (Choice par6: getCategoryById(categories, 6).getPartitions())
                        {
                            if (isNeitherErrorNorSingle(par6.getProperty().getName()) && isNeitherErrorNorSingle(par5.getProperty().getName()) && isNeitherErrorNorSingle(par4.getProperty().getName()) && isNeitherErrorNorSingle(par3.getProperty().getName()) && isNeitherErrorNorSingle(par2.getProperty().getName()) && isNeitherErrorNorSingle(par1.getProperty().getName()) && isNeitherErrorNorSingle(par0.getProperty().getName()))
                            {
                                List<Choice> combinationPars = new ArrayList<Choice>();
                                combinationPars.add(par0);
                                combinationPars.add(par1);
                                combinationPars.add(par2);
                                combinationPars.add(par3);
                                combinationPars.add(par4);
                                combinationPars.add(par5);
                                combinationPars.add(par6);
                                if (isAValidCombination(combinationPars))
                                {
                                    System.out.println(par0.getName() + " - " + par1.getName() + " - " + par2.getName() + " - " + par3.getName() + " - " + par4.getName() + " - " + par5.getName() + " - " + par6.getName());
                                    System.out.println(par0.getPosition() + " + " + par1.getPosition() + " + " + par2.getPosition() + " + " + par3.getPosition() + " + " + par4.getPosition() + " + " + par5.getPosition() + " + " + par6.getPosition());
                                    // write to cell
                                    int subcolumn = 2;
                                    for (int m = 0; m < 7; m++)
                                    {
                                        if (m != subcolumn)
                                        {
                                            Label subcolumn, row, combinationPars.get(m).getName(), cf);
                                            s.addCell(l);
                                            testFrame += combinationPars.get(m).getPosition() + " . " + subcolumn++;
                                        }
                                        // test frame
                                        row, testFrame, cf);
                                        testCaseNo, String.valueOf(testCaseNo), cf);
                                    }
                                    count++;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
public List<List<Choice>> createCombination(int n, List<List<Choice>> coms) {
    List<List<Choice>> combinations = null;
    if (n == 0) return null;
    else {
        return null;
    }
}

private boolean isAValidCombination(Collection<Choice> pars) {
    boolean flag = false;
    int countTrue = 0;
    for (Choice par : pars) {
        System.out.println(par.getName());
        Collection<String> cons = par.getConditions();
        if (cons.size() > 0) {
            int subcountTrue = 0;
            for (String con : cons) {
                if (isExisting(pars, con)) subcountTrue++;
            }
            if (subcountTrue == cons.size()) countTrue++;
            subcountTrue = 0;
        } else {
            countTrue++;
        }
    }
    if (countTrue == pars.size()) return true;
    else return false;
}

public boolean isExisting(Collection<Choice> partitions, String property) {
    for (Choice par : partitions) {
        for (Property pro : par.getProperties()) {
            if (pro.getName().equals(property)) {
                return true;
            }
        }
    }
    return false;
}

public Choice getValidPartitionFromCategory(Category cat) {
    for (Choice par : cat.getPartitions()) {
        if (!(par.getProperty().equals(ERROR) && par.getProperty().equals(SINGLE))) {
            return par;
        }
    }
}
public Category getCategoryById(Collection<Category> categories, int i) {
    int k = 0;
    for (Category cat : categories) {
        if (k == i) return cat;
        k++;
    }
    return null;
}

public boolean isNeitherErrorNorSingle(String s) {
    if (!s.trim().equals(ERROR) && !s.trim().equals(SINGLE)) {
        return true;
    } else {
        return false;
    }
}

public Choice findPartitionByName(String partitionName, Collection<Category> categories) {
    for (Category cat : categories) {
        for (Choice par : cat.getPartitions()) {
            if (par.getName().equals(partitionName)) return par;
        }
    }
    return null;
}

public Choice findPartitionByProperty(String property, Collection<Category> categories) {
    for (Category cat : categories) {
        for (Choice par : cat.getPartitions()) {
            if (par.getProperty().getName().equals(property)) return par;
        }
    }
    return null;
}

public Collection<Choice> findPartitionsByProperty(String property, Collection<Category> categories) {
    Collection<Choice> partitions = new ArrayList<Choice>();
    for (Category cat : categories) {
        for (Choice par : cat.getPartitions()) {
            if (par.getProperty().getName().equals(property)) {
                partitions.add(par);
                // System.out.println(par.getName());
            }
        }
    }
    return partitions;
}

public String getCondition(Collection<String> cons, int i) {
    cons.iterator();
    return null;
}

7.2 Source code being tested

Source code of the module can be downloaded at http://dhis2.org/downloads or check out using launch pad by invoking bzr branch lp:dhis2