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

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

ABSTRACT

Keywords:

1 INTRODUCTION

1.1 Scope of project

This project aims to apply knowledge earned from the Software Verification and Validation UiO Spring 2010 to a practical case: the DHIS2 software, a tool for supporting data collection and processing targeted at health care sector. Software testing is widely accepted as one of the most important activities to ensure quality (). 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 methodology to test against those applications becomes more and more urgent.

1.2 Goals of project

DHIS2 has been adopted and used in a number of countries and in some places like India and Serra Leon, it has become the national systems for public health statistics. Developed by University of Oslo, Informatics Department, and followed Open Source Software (OSS) tradition, DHIS2 get involved a heterogeneous team with members coming from different domains: informatics, public health, spreading over continents. From the beginning, testing was only done at the unit level by developers who build that functions, using JUnit framework. Functional testing is rarely implemented or in a very ad hoc

manner. There are neither documents related to test activity nor ideas on how the system is tested. Just 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. And this step costs lots of money.

What can be referred from this story is the need of more systematic approaches for quality assurance in DHIS2 software. Hence, this study has two objectives: a) to develop a framework in order to build test suites 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, and tools are discussed, in line with a literature review on black box testing in generally and category partition in particular.

2 BACKGROUND

As previously discussed, category partition, among many other methods of black box testing approach was selected in order to carry out this study. The following discussion will explain the 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 requiring knowledge on how the system was built, and 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?

There are many techniques for black box testing that have been developed (Ostrand and Balcer 1988). For example, the choice relation framework,[2] the classification-tree methodology,[3] domain testing[4] 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 mentions boundary analysis and logical functions as the other techniques for black box testing.

Poon (2008) considers all the techniques are similar in term of steps in which tests case are generated: 1) identify categories and choices 2) combine valid choices 3) build test cases based on valid combination. However, among them, category partition method (CPM) 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). 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. However, 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[1].

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.

Among other techniques, CPM has become a very popular methods, widely adopted, studied, and improved by vendors and researchers[2]. 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 re-consideration of traditional methods for object-oriented software, arguing that CPM can be effective to uncover faults in object-oriented software. More recently, Lionel et al (2009) apply machine-learning approach to refine test specification and test suites in CPM.

2.3 Why test frame generator?

Large body of research centered on 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

However, according to my knowledge, there has not yet been a tool for transferring categories/partitions into test frames though it is considered as an easy-to-automate step. This tool is supposed to read Test Specification, i.e. categories/partitions/constraints, from an input file (text, xml, or probably database), and then combine them into 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 partitions. The reason for this program can be easily found in the literature: system specification is written in natural language, selecting categories/partitions is dependent very much on experience of test engineers.

As this step of categorizing and partitioning is always problematic, the need for refactoring categories/partitions/constraints in order to build up a new list of combination becomes obvious and urgent.

---

[1] http://benderrbt.com/bendersoftware.htm#over

[2] Indeed, Ostrand and Balcer 1988 has been cited more than 100 times

3 DESIGN OF THE CASE STUDY

3.1 System under test (SUT) description

District Health Information System version 2 (DHIS2) is an open source software based on popular Java Enterprise frameworks such as Spring, Hibernate, Struts2. Developed by University of Oslo and other developers within the umbrella of Health Information System Programme, a program that aims at strengthening health information system in developing countries, DHIS2 is a system for data collection, processing, and analysis. The core data model of DHIS2 is presented as follows:

- Organization Unit: could be a country, province, district, or a facility like hospital, clinic
- Data Element: number of children under 5, number of children who have BCG, number of pregnant women, number of HIV cases…
- Period: could be year, quarter, or month.
- Data Value: a number present a data element value of one organization unit within a particular period.

Architecture:

DHIS2 employs modular and layer design. In overall, it is separated into 3 layers:

- 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: adopts MVC (Model View Controller) design by using Struts2. The web layer is structured by modules (there are currently almost 20 web modules), which communicates through a module called web-portal.

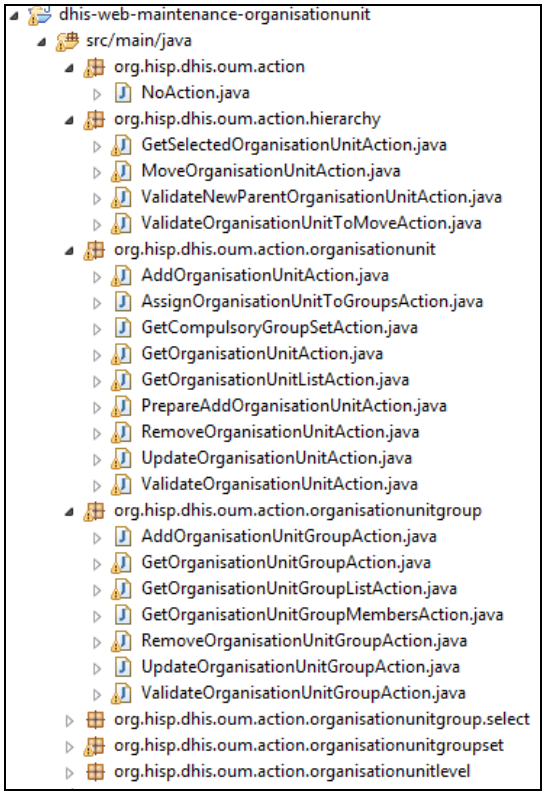
Currently, DHIS2 has only unit tests for service layer, there are any unit tests at web layer. The functional testing (black box) is done manually via web-based user interface with a list of test cases selected randomly. This strategy appears inappropriate when deploying the system in a large scale, i.e. in India.

The core module of DHIS2 provides basic functionalities to setup a reporting system, as described above. It has CRUD operations for data element, organization units, data set (a list of data element). DHIS2 has more than 50 modules consisting thousands of classes. Therefore, in this assignment, I plan to test only one among them. The SUT hence is now discussed.

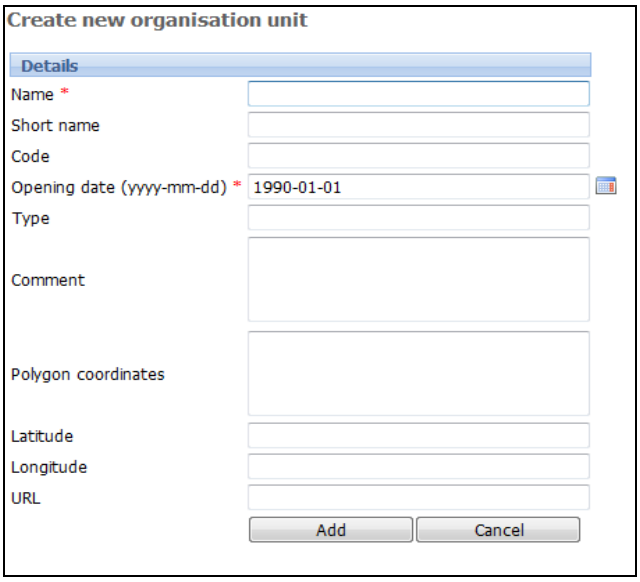
3.2 The dhis-web-maintainance-organisationunit module

This module is one of module in the web group providing user interface for end users to interact with the system. Main classes in this module are Action classes that extend ActionSupport class provided by Struts2 MVC (Model View Controller) framework. The action classes receive inputs from users, process them via execute methods, and return appropriate outputs.

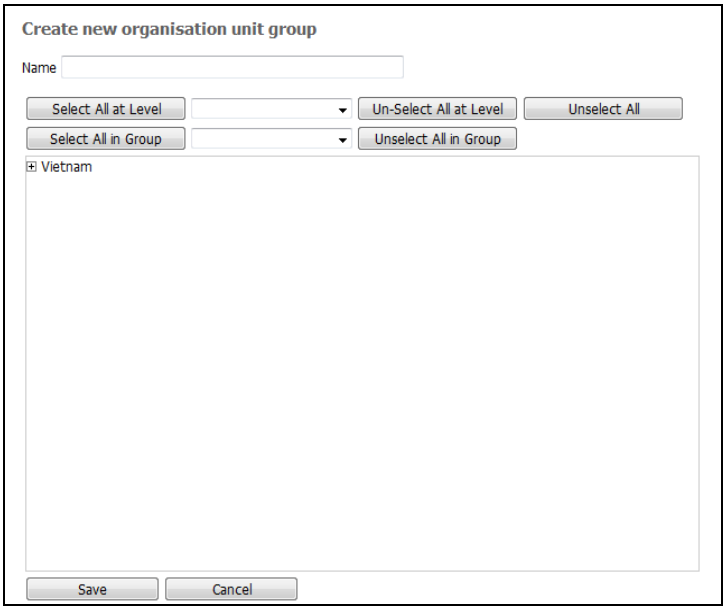
This module is responsible for tasks related to organization units, unit group, and unit group set such as basic CRUD, assigning new parent to an organization unit (moving).



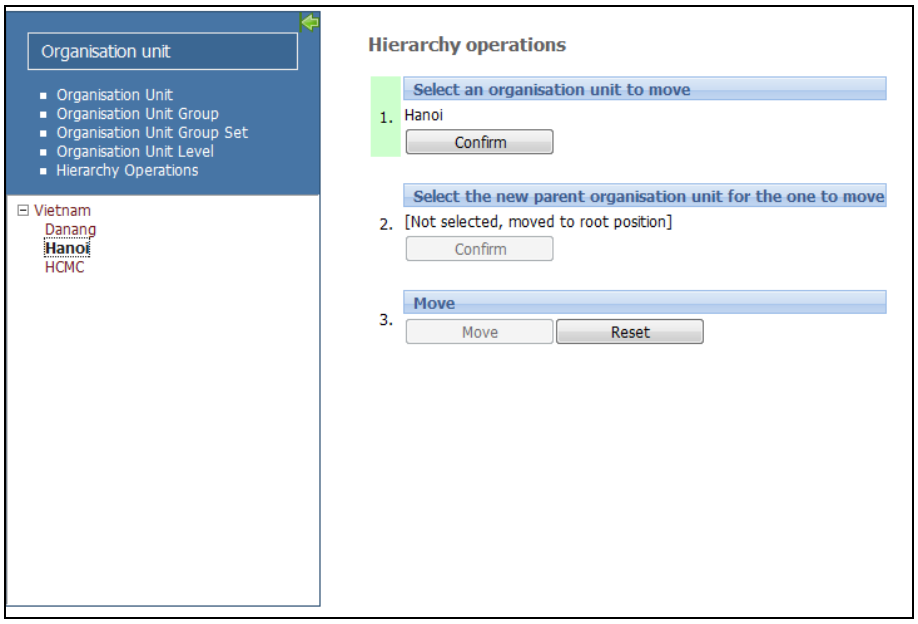
The form to add new organization unit:



To add new unit group



To move a unit



3.3 The specification of the module

As an OSS, DHIS2 has not been followed traditional software process hence documentation of specification is very limited. Expected functionalities, bugs, defects, milestones are discussed broadly through mailing list (dhis2-dev), bug tracking (launchpad), and its website. For the purpose of this assignment, a detailed specification of the module is provided as the following:

Specification	Name of functionalities	Description
Organisation unit		
	Add new organisation	The home page of the module show a list of units. User click Add new organisation 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 requirement in the adding unit case.

	Delete an organisation 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 a 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 to be moved with it.

Table 1: System specification

Therefore, a more systematic and well organized functional testing is needed urgently to make users confident on using the system and reduce manual time-consuming test. I will go for black box testing (BBT) strategy at first by employing Category Partition (CP), due to the following reasons:

- In my case, BBT is appropriate to answer the question "Does the implementation correctly implement the functionality as per the given system specifications?"[3]
- There are also other techniques in BBT such as Week/Strong Equivalence Class Partition (EP), Boundary Value Analysis (BVA) but CP is an extension and combination of the two, helping to reduce dramatically number of test cases needed in the case of EP and BVA.
- Two "best" techniques among those in BBT are selected to compare their efficiency in a particular case of web applications, i.e. DHIS2 software.

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)

The test specification language was first proposed by Ostrand and Balcer (1988). According to this language, categories and partitions are put as a list. Properties of each partition are specified by the syntax [property X, Y] with X and Y are two properties separated by a comma. The two other annotations [error] and [single] to indicate two cases of special partitions which can help to reduce the number of combination. The [error] partition means that this partition could not produce a valid input while the [single] partition indicates that the partition need to combine to only one partition of other categories. The notation [if X, Y,…] applies for partitions in which they are only able to be combined with other partitions which have such properties. In their paper, Ostrand and Balcer (1988) also give an example on how it is used for a particular case.

[3] Blackbox testing slides of the course slide 2.
http://www.uio.no/studier/emner/matnat/ifi/INF4290/v10/undervisningsmateriale/INF4290-BBT.pdf

Though the TSL proposed by Ostrand and Balcer (1988) seems very simple and easy to read. It has several limitations. First, the character [] used for describing properties of partitions does take longer to by typed. Second, as specified as a flat document, it increases difficulty for automation, i.e. not easy to be read by a computer program.

3.4.2 A revised TSL

Therefore, in this paper, 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 which separates category, partition, property, and condition into different columns. With this new approach, we could save certain amount of time in typing the special characters [,], and if while still keeping readability of the language . The modified TSL are summarized in the following table:

STT	Change	
1	Employ column-oriented approach	
2	Remove the open and close bracket [and]	
3	Remove the keyword if	

3.4.3 Using Excel file as input/output

There are different alternatives for input formats such as hierarchy structure xml file, comma separated value (csv) file, relational database structure, and excel. Among the alternatives, Excel is selected 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

- Data model:

The system consists of the following classes:

a. Category

b. Partition

c. Property

d. CPGenerator

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

- Algorithm: the CPGenerator class has several methods and the algorithm operates at the following steps:

a. The algorithm for the test frame generator can be specified in pseudo code as follows:

b. Reading the input Excel file

c. Building a data structure for the test specification of the Excel file

d. Create test frame with error annotation

e. Create test frame with single annotation

f. Create test frame remaining

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

3.5 Initial results

In this section, I try to apply the tool developed in the previous section to test the SUT described in the case study – the organization unit management module within DHIS2 software. The following steps were done:

- Building the categories, partitions, and constraints using the revised TSL (Excel file as input)
- Applying the tool for the case to build the test set (Excel file as output)
- Run the test set

3.5.1 Organization unit functionalities

Adding:

Test specification

Categories	**Partitions**	**Properties**	**Conditions**
openingdate			
	null	error	
	valid		
	notvalid	error	
closingdate			
	null	single	
	valid and >= openingdate		
	valid and < openingdate	error	
	notvalid	error	
selectedorg			

	null	single	
	notnull		
environment			
	name not exist		nameok
	shortname not exist		shortnameok
	name exist		nameok
	shortname exist		shortnameok

Test frames

No	Test frame ID	name	shortname	openingdate	selectedorg	environment	Expected output	Pass/Fail
1		length=0					Error message on invalid input	P
2		length=1					Error message on invalid input	P
3		length>255					Error message on invalid input	F
4			length=0				Error message on invalid input	P
5			length>25				Error message on invalid input	P
6				null			Error message on invalid	P

							input	
7				notvalid			Error message on invalid input	P
8	3.2.2.1.1.1.1.	length=2-255	length=2-25	valid	null	name not exist	Add successfully a root orgunit	P
9	3.2.2.2.1.1.1.	length=2-255	length=2-25	valid	notnull	name not exist	Add successfully a non-root orgunit	P
10	3.2.2.2.2.1.1.	length=2-255	length=2-25	valid	notnull	shortname not exist	Add successfully a non-root orgunit	P
11	3.2.2.2.3.1.1.	length=2-255	length=2-25	valid	notnull	name exist	Error message on existing name	P
12	3.2.2.2.4.1.1.	length=2-255	length=2-25	valid	notnull	shortname exist	Error message on existing shortname	P

Editing: the selected organization input is not available

Test specification

Categories	**Partitions**	**Properties**	**Conditions**
name			
	length=0	error	
	length=1	error	
	length=2-255	nameok	
	length>255	error	
shortname			

	length=0	error	
	length=2-25	shortnameok	
	length>25	error	
openingdate			
	null	error	
	valid		
	notvalid	error	
closingdate			
	null	single	
	valid and >= openingdate		
	valid and < openingdate	error	
	notvalid	error	
environment			
	name not exist		nameok
	shortname not exist		shortnameok
	name exist		nameok
	shortname exist		shortnameok

Test frames

No	**Test frame ID**	**name**	**shortname**	**openingdate**	**closingdate**	**environment**	**Expected output**	**Pass/Fail**
1		length=0					Error message on invalid input	P
2		length=1					Error message on invalid input	P
3		length>255					Error message on invalid input	F
4			length=0				Error message on invalid	P

							input	
5			length>25				Error message on invalid input	P
6				null			Error message on invalid input	P
7				notvalid			Error message on invalid input	P
8	3.2.2.1.1.1.1.	length=2-255	length=2-25	valid	null	name not exist	Update successfully orgunit	P
9					valid and < openingdate		Error message on invalid input	P
10					notvalid		Error message on invalid input	P
11	3.2.2.2.1.1.1.	length=2-255	length=2-25	valid	valid and >= openingdate	name not exist	Update successfully orgunit	P
12	3.2.2.2.2.1.1.	length=2-255	length=2-25	valid	valid and >= openingdate	shortname not exist	Update successfully orgunit	P
13	3.2.2.2.3.1.1.	length=2-255	length=2-25	valid	valid and >= openingdate	name exist	Error message on existing name	P
14	3.2.2.2.4.1.1.	length=2-255	length=2-25	valid	valid and >= openingdate	shortname exist	Error message on existing	P

							name	

Deleting

Test specification

Categories	**Partitions**	**Properties**	**Conditions**
environment			
	orgunit contains a child		
	orgunit contains no child		

Test frames

No	**Test frame ID**	**environment**	**Expected output**	**Pass/Fail**
1	1.1.1.1.1.1.1.	orgunit contains a child	Fail to delete	
2	2.1.1.1.1.1.1.	orgunit contains no child	Succeed to delete	

3.5.2 Organization unit group

Adding

Test specification

Categories	**Partitions**	**Properties**	**Conditions**
name			
	length=0	error	
	length=1	error	
	length=2-255		
	length>255	error	
list of orgunit			
	0 orgunit		
	1 orgunit		
	2 orgunit		

environment			
	name does not exist		
	name exists	single	

Test frames

No	Test frame ID	name	list of orgunit	environment	Expected output	Pass/Fail
1		length=0			Error message on invalid input	P
2		length=1			Error message on invalid input	P
3		length>255			Error message on invalid input	F
4	3.1.1.1.1.1.1.	length=2-255	0 orgunit	name does not exist	Error successfully group with 0 orgunit	P
5	3.3.1.1.1.1.1.	length=2-255	2 orgunit	name does not exist	Error successfully group with 2 orgunits	P
6	3.2.1.1.1.1.1.	length=2-255	1 orgunit	name does not exist	Error successfully group with 1 orgunit	P
7	3.2.2.1.1.1.1.	length=2-255	1 orgunit	name exists	Error message on existing name	P

Editing

Test specification

Categories	**Partitions**	**Properties**	**Conditions**
name			
	length=0	error	
	length=1	error	
	length=2-255		
	length>255	error	
list of orgunit			
	0 orgunit		
	1 orgunit		
	2 orgunit		
environment			
	name exists	single	
	name does not exist		

Test frames

No	**Test frame ID**	**name**	**list of orgunit**	**environment**	**Expected output**	**Pass/Fail**
1		length=0			Error message on invalid input	P
2		length=1			Error message on invalid input	P
3		length>255			Error message on invalid input	F
4	3.1.1.1.1.1.1.	length=2-255	0 orgunit	name exists	Error message on existing name	P

5	3.1.2.1.1.1.1.	length=2-255	0 orgunit	name does not exist	Update successfully group with 0 orgunit	P
6	3.2.2.1.1.1.1.	length=2-255	1 orgunit	name does not exist	Update successfully group with 1 orgunit	P
7	3.3.2.1.1.1.1.	length=2-255	2 orgunit	name does not exist	Update successfully group with 2 orgunit	P

Deleting

Test specification

Categories	**Partitions**	**Properties**	**Conditions**
environment			
	group has 1 orgunit		
	group has 0 orgunit		

Test frame

No	**Test frame ID**	**environment**	**Expected output**	**Pass/Fail**
1	1.1.1.1.1.1.1.	group has 1 orgunit	Fail to delete	P
2	2.1.1.1.1.1.1.	group has 0 orgunit	Succeed to delete	P

3.5.3 Group set

Adding

Test specification

Categories	**Partitions**	**Properties**	**Conditions**

name			
	length=0	error	
	length=1	error	
	length=2-255		
	length>255	error	
description			
	length=0	error	
	length=1	error	
	length=2-255		
	length>255	error	
compulsory			
	Yes		
	No		
list of group			
	1 group		
	2 groups	single	
	0 group		
environment			
	group set name exist	single	
	group set name does not exist		

Test frame

		name	**description**	**compulsory**	**list of group**	**environment**	**Expected output**	**Pass/Fail**
1		length=0					Error message on invalid input	P
2		length=1					Error message on invalid input	P
3		length>255					Error message on invalid input	P

4			length=0				Error message on invalid input	P
5			length=1				Error message on invalid input	P
6			length>255				Error message on invalid input	F
7	3.3.1.2.2.1.1.	length=2-255	length=2-255	Yes	2 groups	group set name does not exist	Add successfully group set with 2 groups	P
8	3.3.1.1.1.1.1.	length=2-255	length=2-255	Yes	1 group	group set name exist	Error message on exisiting name	P
9	3.3.1.1.2.1.1.	length=2-255	length=2-255	Yes	1 group	group set name does not exist	Add successfully group set with 1 groups	P
10	3.3.1.3.2.1.1.	length=2-255	length=2-255	Yes	0 group	group set name does not exist	Error message when compulsory is Yes but no group is selected	P
11	3.3.2.1.2.1.1.	length=2-255	length=2-255	No	1 group	group set name does not exist	Add successfully group set with 1	P

							groups	
12	3.3.2.3.2.1.1.	length=2-255	length=2-255	No	0 group	group set name does not exist	Add successfully group set with 0 groups	P

Editing

Test specification

Categories	**Partitions**	**Properties**	**Conditions**
name			
	length=0	error	
	length=1	error	
	length=2-255		
	length>255	error	
description			
	length=0	error	
	length=1	error	
	length=2-255		
	length>255	error	
compulsory			
	Yes		
	No		
list of group			
	1 group		
	2 groups	single	
	0 group		
environment			
	group set name exist	single	
	group set name does not exist		

Test frame

		name	description	compulsory	list of group	environment	Expected output	Pass/Fail
1		length=0					Error message on invalid input	P
2		length=1					Error message on invalid input	P
3		length>255					Error message on invalid input	F
4			length=0				Error message on invalid input	P
5			length=1				Error message on invalid input	P
6			length>255				Error message on invalid input	P
7	3.3.1.2.2.1.1.	length=2-255	length=2-255	Yes	2 groups	group set name does not exist	Add successfully group set with 2 groups	P
8	3.3.1.1.1.1.1.	length=2-255	length=2-255	Yes	1 group	group set name exist	Error message on existing name	P

9	3.3.1.1.2.1.1.	length=2-255	length=2-255	Yes	1 group	group set name does not exist	Add successfully group set with 1 groups	P
10	3.3.1.3.2.1.1.	length=2-255	length=2-255	Yes	0 group	group set name does not exist	Error message when compulsory is Yes but no group is selected	P
11	3.3.2.1.2.1.1.	length=2-255	length=2-255	No	1 group	group set name does not exist	Add successfully group set with 1 groups	P
12	3.3.2.3.2.1.1.	length=2-255	length=2-255	No	0 group	group set name does not exist	Add successfully group set with 0 groups	P

Deleting

Test specification

Categories	**Partitions**	**Properties**	**Conditions**
environment			
	group set contains a child		
	group set contains no child		

Test frame

		environment	Expected output	Pass/Fail
1	1.1.1.1.1.1.1.	group set contains a child	Fail to delete	P

2	2.1.1.1.1.1.1.	group set contains no child	Succeed to delete	P

3.5.4 Hierarchy operators

Test specification

Categories	**Partitions**	**Properties**	**Conditions**
orgunit to be moved			
	non-root orgunit	nonroot	
	root orgunit		
target orgunit			
	same as the moving orgunit		
	child of the moving orgunit		
	parent of the moving orgunit		nonroot
	brother of the moving orgunit		

Test frame

		orgunit to be moved	**target orgunit**	**Expected output**	**Pass/Fail**
1	1.1.1.1.1.1.1.	non-root orgunit	same as the moving orgunit	Error message about one group can not be moved to itself	P
2	1.2.1.1.1.1.1.	non-root orgunit	child of the moving orgunit	Error message about one group can not be moved to its children	P
3	1.3.1.1.1.1.1.	non-root orgunit	parent of the moving orgunit	Successful but unchanged	P
4	1.4.1.1.1.1.1.	non-root orgunit	brother of the moving orgunit	Successful move	P
5	2.1.1.1.1.1.1.	root orgunit	same as the moving orgunit	Error message about one group can not be moved	P

				to itself	
6	2.2.1.1.1.1.1.	root orgunit	child of the moving orgunit	Error message about one group can not be moved to its children	P
7	2.4.1.1.1.1.1.	root orgunit	brother of the moving orgunit	Successful move	P

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 in term of time constraint. It is also very much smaller compared to the number of test cases when not using the method. The exhausted test approach is an impossible task. For example, in the adding organization unit functionality, for only input value of property Organization unit name, there are millions combinations.

Also, the equivalence partition method also result 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.

4.1.2 Effectiveness of the tool:

Though there has not been a research carried out to assess the effectiveness of the tool, from the case study, the tool has helped to discover and correct several mistakes when defining categories and partitions.

For example, data in the following table shows how the tool can help to quickly fix the test frame table if testers make a mistake in building test specification . This table is based on the hierarchy operator functionality.

In the specification **Before,** I made mistake when forgetting to define a constraint for orgunit to be moved and 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.

	Specification				Test frames		
Before	**Categories**	**Partitions**	**Properties**	**Conditions**	**No**	**orgunit to be moved**	**target orgunit**
	orgunit to be				1	non-root orgunit	same as the moving orgunit

	moved			
		non-root orgunit		
		root orgunit		
	target orgunit			
		same as the moving orgunit		
		child of the moving orgunit		
		parent of the moving orgunit		
		brother of the moving orgunit		

2	non-root orgunit	child of the moving orgunit
3	non-root orgunit	parent of the moving orgunit
4	non-root orgunit	brother of the moving orgunit
5	root orgunit	same as the moving orgunit
6	root orgunit	child of the moving orgunit
7	**root orgunit**	**parent of the moving orgunit**
8	root orgunit	brother of the moving orgunit

After

Categories	**Partitions**	**Properties**	**Conditions**
orgunit to be moved			
	non-root orgunit	**nonroot**	
	root orgunit		
target orgunit			
	same as the moving orgunit		
	child of the moving orgunit		
	parent of the moving orgunit		**nonroot**
	brother of the moving orgunit		

No	**orgunit to be moved**	**target orgunit**
1	non-root orgunit	same as the moving orgunit
2	non-root orgunit	child of the moving orgunit
3	non-root orgunit	parent of the moving orgunit
4	non-root orgunit	brother of the moving orgunit
5	root orgunit	same as the moving orgunit
6	root orgunit	child of the moving orgunit
7	root orgunit	brother of the moving orgunit

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

Specification

Before

Categories	**Partitions**	**Properties**	**Conditions**
name			
	length=0	error	
	length=1	error	
	length=2-255		
	length>255	error	
list of orgunit			
	0 orgunit	single	
	1 orgunit		
	2 orgunit	single	
environment			
	name exists		
	name does not exist		

Test frames

	name	**list of orgunit**	**environment**
1	length=0		
2	length=1		
3	length>255		
4	length=2-255	0 orgunit	name exists
5	length=2-255	2 orgunit	name exists
6	length=2-255	1 orgunit	name exists
7	length=2-255	1 orgunit	name does not exist

After

Categories	**Partitions**	**Properties**	**Conditions**
name			
	length=0	error	
	length=1	error	
	length=2-255		
	length>255	error	
list of orgunit			
	0 orgunit		
	1 orgunit		

No	**name**	**list of orgunit**	**environment**
1	length=0		
2	length=1		
3	length>255		
4	length=2-255	0 orgunit	name exists
5	length=2-255	0 orgunit	name does not exist
6	length=2-255	1 orgunit	name does not exist

		2 orgunit			7	length=2-255	2 orgunit	name does not exist
	environment							
		name exists						
		name does not exist	single					

The table shows that the **Before** Specification led to three redundant test frames which are marked as red color. When the name of group is already existing in the system, test cases consisting the existing name with three different options of orgunit list are not useful at all.

4.1.3 Results of running the test set:

Manually running the test set described previously produced the results with six failed test cases. All of them belong to the same category that is related to the length of input value.

The defect rate = 6/87 *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 most of implementation sites, this module was used exclusively by the HISP team and the organization unit hierarchies for each place were given to users and "factory setting". If the module had been used by the end users, probably these bugs could have been discovered. For other more complicated modules, the failure rates could be higher.
- Applying a systematic approach to testing such as CPM can increase the confidence in order to release the product. It also helps to estimate time for testing, hence, making better project planning.

4.2 Summary of important results

Here are several points this assignment

- 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 seems difficult to apply or incapable to support the testing process, 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 and partitions 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 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 risk. 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[4]. 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 the assignment can make. Is it to find as many bug as possible in the SUT? 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?

5.2 Limitation & future research

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

- The algorithm need to be extended in order to address the issues of limitation in the number of inputs. Currently only maximum seven inputs are allowed.
- The process which converts test frames to test cases can be difficult to be automated as it is based on narrative language. However, in each test case, input and environmental values can could given by testers hence there will be more chances for automation from the test cases.
- More research needs to be carried out to collect feedbacks from users to compare the productivity and the quality of the test sets between two groups of users: using the tool and not using the tool.

[4] Indeed, there is tool called BenderRBT for cause effect graph but it is not OSS (http://benderrbt.com/)

6 REFERENCES

Book:

A. Mathur, Foundations of Software Testing, Pearson Education, 2008

M. Pezze and M. Young, Software Analysis and Software Testing, Wiley, 2007

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

Grottke, M.; K.S. Trivedi; "Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate," *IEEE Computer*, vol. 40, no. 2, 2007, p. 107-109

T.Y. Chen; P.-L. Poon; T.H. Tse; "A Choice Relation Framework for Supporting Category-Partition Test Case Generation," *IEEE Transactions on Software Engineering*, vol. 29, no. 7, 2003, p. 577-593

T.Y. Chen; P.-L. Poon; T.H. Tse; "An Integrated Classification-tree Methodology for Test Case Generation," *International Journal of Software Engineering and Knowledge Engineering*, vol. 10, no. 6, 2000, p. 647-679.

Grochtmann, M.; K. Grimm; "Classification Trees for Partition Testing," *Software Testing, Verification and Reliability*, vol. 3, no. 2, 1993, p. 63-82.

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

Krishnan, R.; S.M. Krishna; P.S. Nandhan; "Combinatorial Testing: Learnings From Our Experience," *ACM SIGSOFT Software Engineering Notes*, vol. 32, no. 3, 2007, p. 1-8

Gary E. MogyorodiThe Journal for Software Testing Professionals, March, 2003: Requirements-Based Testing: Cause-Effect Graphing

N. Amla and P. Ammann, "Using Z Specifications in Category-Partition Testing," Systems Integrity, Software Safety, and Process Security: Building the Right System Right: Proc. Seventh Ann. IEEE Conf. Computer Assurance (COMPASS '92), pp. 3-10, 1992.

P. Ammann and J. Offutt, “Using Formal Methods to Derive Test Frames in Category-Partition Testing,” Safety, Reliability, Fault Tolerance, Concurrency, and Real Time Security: Proc. Ninth Ann. Conf. Computer Assurance (COMPASS '94), pp. 69-79, 1994.

M. Grochtmann and K. Grimm, “Classification Trees for Partition Testing,” Software Testing, Verification and Reliability, vol. 3, no. 2, pp. 63-82, 1993.

A.J. Offutt and A. Irvine, “Testing Object-Oriented Software Using the Category-Partition Method,” Proc. 17th Int'l Conf. Technology of Object-Oriented Languages and Systems (TOOLS 17), pp. 293-304, 1995.

7 APPENDICES

Appendices should provide all detailed, relevant information that will allow me to better assess what has been done in the project.

Examples:

☐Detailed test models and requirements

☐Test driver code (must be commented), test stubs

☐Detailed test results

☐If possible, source code being tested

☐ UML models