Research on Automated Software Test Case Generation

Daisen Wei
College of computer science and technology, Shandong University, Jinan, China
Inspur Genersoft Co., Ltd. Jinan, China
E_mail: weids@inspur.com

Longye Tang
Inspur Genersoft Co., Ltd. Jinan, China
E_mail:tangly@inspur.com

Xueqing Li
College of computer science and technology, Shandong University, Jinan, China
E_mail: xqli@sdu.edu.cn

Ling Shang
Lifl, University of Science and Technology of Lille, Lille, France
E_mail: ling.shang@lifl.fr

***Abstract*—To improve the efficiency of software testing, a model-driven method is proposed to automatically generate test cases from UML design model. In it, PITCs (platform-independent test cases) are generated first from a UML design model. And then, according to the predefined rules, a process is implemented to transform PITCs into the corresponding PSTCs (platform-specific test cases). The experiment and comparison had showed that the method proposed in this paper was easier to be understood and implemented by users to generate test cases than the ones existed.**

***Index Terms*—software testing, test case, PITC, PSTC, transformation rule**

I. Introduction

Testing is one of key steps in software development and even runs through the whole software lifecycle. And now, it is regarded as one of the most important and effective ways to improve the quality of software by trying to find the potential faults that may exist in source codes. The first step of implementing it is to design and generate test cases, a set of data generally including input and the expected output that satisfy the design or testing requirements of SUT (system under test).

In recent decades, UML has been one of the most popular design tools widely used in some key subprocesses of software lifecycle, especially design and testing. So, UML based testing has already been paid more attention by researchers from both academia and industry. UML models, however, are generally in the form of diagrams, so it is very hard to directly generate test cases from them. One approach adopted to deal with it was to create the corresponding test models from UML design ones, from which test cases were generated. A specification, UML2.0 test profile, had also been released by OMG in 2004[1] to support this model-based testing. And on the basis of it and the methodology of MDD (model driven development), model-riven testing were also widely researched [2]. One of the advantages of this method is to provide a good way to automatically generate test cases through model transformation.

Model-riven testing, however, is also a challenge for testers due to some reasons, one of which is that how the input space of SUT is to be defined and then from which the appropriate testing data are selected to form test cases. This is one of the important factors to influence that whether test cases can automatically be generated or not. And this problem can also cause that a test case would usually be defined as a form, in most methods existed, being different from that of the executable one with the real input/expected output data used in practical testing. Obviously, all these can make the process of test case generation low efficiency and time consuming. So, research on automated test case generation from UML models is necessary and very worthy of doing. More attention had also been paid to it in academia and industry and some achievements had been obtained in recent years [3].

In this paper, one model-driven method was proposed to generate software test cases, the executable ones, from a UML design model. And state diagram was selected in a case given in section V below. The idea of this method

Manuscript received January 20, 2014; revised May 22, 2014; accepted May 23, 2014.

originated from MDD, in which object source codes were automatically generated by model transformation from UML design models. The basic process to implement it was that PITCs were generated from a UML design model, PIM (platform independent model), and then data mapping rules from PITC to PSTC were defined to guide the PSTCs generation from the corresponding PITCs.

The remainder of this paper is organized as follows. In section II, some concepts such as PITC, PSTC and transformation from PITC to PSTC etc. are defined. In section III, the processes including PITC generation, PITC-to-PSTC transformation and PSTC generation are described respectively. In section IV, a case is given to show the whole PITC-to-PSTC process. In section V, the method is analyzed and compared to some related ones from two perspectives. And the conclusions are given in section VI.

II. Some Concepts

A test case in software engineering is a set of conditions or variables under which a tester will determine whether an application, software system or one of its features is working as it was originally established for it to do. In this paper, a software test case is defined as follows:

Definition 1: Test Case (TC). A test case is a 3-tuples: (Id, InitState, Data), where: (1) the "Id" is an unique and numbered string assigned to each test case, and (2) the "InitState" is the current state of SUT, followed by the execution of the test case selected, and (3) the "Data" is a set in the form of {<ini, eouti>} in which the "ini" represents the input and the "eouti" the expected output.

Note that: the data pair <ini, eouti> implements an atomic testing step. The "atomic" means that the <ini, eouti> is the minimum input-output data pair, that is, there are no other input or output data between the "ini" and the "eouti".

Definition 2: Platform-Independent Test Case (PITC). A PITC is a test case generated from system design model, in which all data including parameters' types, values and syntax are not bound to any programming language such as JAVA or a platform specification such as .NET and JSP. For example, the following data string is a PITC being designed to verify the validity of user's identity before he or she tries to login and enter a web system.

(tc1, *login page*, < [*name, password*], *main page of the system*>)

In the PITC tc1, the "login page" means the page loaded for users to login and also represents the current system state before the users' account are entered. The "[name, password]" represents the user's account as input. The "main page of the system" means the loaded page as output, that is, the new system state after the user's name and password are entered and then submitted. As we can see, all data involved in tc1 are platform-independent. In this paper, this type of data is called as platform-independent data (PID), the form of data without definite values in a PIM.

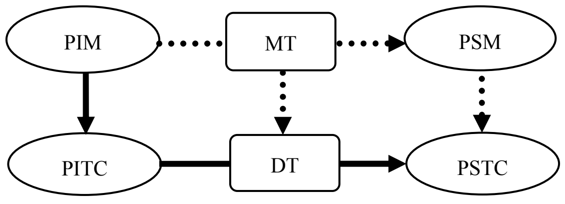
Definition 3: Platform-Specific Test Case (PSTC). A PSTC is the refined version of a PITC, in which all data involved are platform-specific and the syntax of them conforms to a specific programming language or platform specification. The following PSTC tc11 corresponding to the above PITC tc1 is given as follows:

(tc11, UserLogin.jsp, < ["administrator","12@abMN67"], default.jsp >)

Note that: the syntax of tc11 complies with the object specification JSP. In it, the NO. of test case, tc11, can be changed as required. The "*UserLogin.jsp*" is a JSP page that means the concrete login page of SUT. The pair "*['administrator','12@abMN67']*" represents the login account including the user's name and password. And the "default.jsp" represents the main page of SUT as expected output after a user's account is verified to be true.

From the definition 4 and 5 above, it can be seen that a PSTC can be executed manually but a PITC cannot. In this example, the syntax of all data in tc11 conforms to the platform specification JSP. The tc11 is also a test case that can be executed manually. And if required, it can further be transformed automatically into a script that can directly be executed by test tools. In this paper, this type of data is called as platform-specific data (PSD), the form of ones with definite values in a PSM (platform specific model). In MDD, a PSM is always transformed from a specific PIM.

The figure 1 below shows the test case generation process marked with the directed real lines. And it consists of two subprocesses, one of which is the PITCs generation from PIM and the other is the PSTCs generation through the data transformation from the corresponding PITCs.



MT-Model Transformation
DT-Data Transformation

Figure 1 Model-driven test case generation

Note that: in figure 1, the process "MT" means the model transformation from PIM to PSM, which is implemented according to the predefined rules. And this transformation process involves two subprocesses of refinements, logic structure or syntax and data object, from PIM to PSM. The process "DT" represents the refinement of data objects from PIM to PSM. So, DT is only one part of MT.

The data object is defined as follows:

Definition 4: Data Object (DO). A DO is an object with attributes that appears in test cases for SUT.

In object-oriented methodology, an object usually consists of attributes and methods or functions. If an object appears in a test case, only the values assigned to the corresponding attributes of it are involved generally.

So, the DO is defined here just from the perspective of test cases, that is, only the attributes and their values of an object are focused and used.

And in this paper, a DO has two forms, one of which is PIDO (platform independent data object) and the other PSDO (platform specific data object). The only difference between them is that PIDO is defined or used in PIM and the corresponding PSDO in PSM. Correspondingly, the attributes of a PIDO are named as PIA (platform independent attributes) and the ones of a PSDO as PSA (platform specific attributes). The states of a PIDO are called as PIS (platform independent states) and the ones of a PSDO as PSS (platform specific states). And all these terms are used in the following figure 2 and definition 5.

For instance, in the PITC tc1 given above, two attribute parameters, name and password, together define a DO user account. The same DO user account is described in the form of [name, password] in tc1 and correspondingly, that in the form of [“administerator”,”12@abMN67”] in the PSTC tc11. Obviously, it shows that a PITC and all data objects in it are platform independent and a corresponding PSTC and all the same data objects in it platform specific.

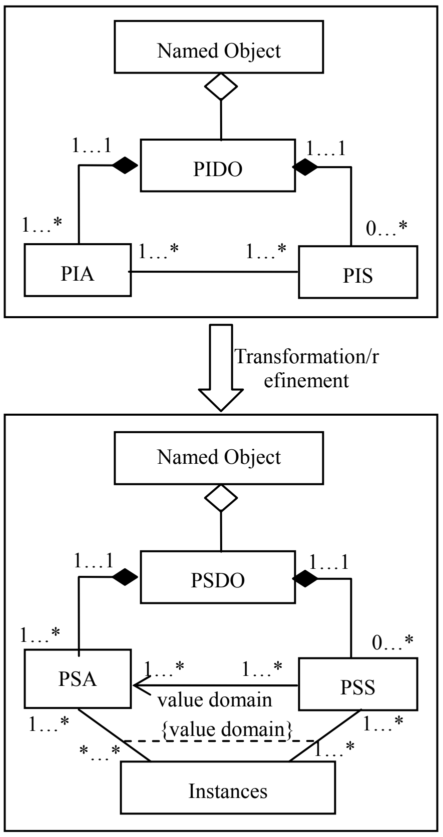


Figure 2 Transformation from PIDO to PSDO

Definition 5: ⓣ. The ⓣ is defined as the operation of transforming a PITC into the corresponding PSTC (s).

Because each attribute variable can be assigned to different values that may represent different states of a data object, the ⓣ implements a one-to-many function. That is, one PITC can be transformed into many PSTCs. One example is that each variable should be assigned at least to two constant values, the valid one accepted by SUT and the invalid one failed in SUT.

The detailed process of transforming a PITC into the PSTC (s) was given in section III(C) below.

In essence, the operation ⓣ implements the process of data refinement between PIM and PSM, in which only constant values from the data space of PSM are assigned to the corresponding attributes of PIDO in PIM. Because the ⓣ does not change the semantic of these attributes, it should keep the property preservation in this transformation from a PITC to the corresponding PSTCs.

For example, the transformation from the PITC tc1 to the PSTC tc11 given above can be correspondingly described as the following table I.

TABLE I

AN EXAMPLE: TRANSFORMATION FROM PIDO TO PSDO

DO	PIDO in the PITC *tc1*	PSDO in the PSTC *tc11*
page	login page	*UserLogin.jsp*
user account	name, password	*“administrator”, ”12@abMN67”*
page	main page of the system	*default.jsp*

III. PITCS AND PSTCS GENERATION

A. Generating the Executable Paths

A test case always corresponds to an executable path in SUT. In this paper, the approach to generating test cases is on the basis of UML design models such as activity and state diagram. So, in order to be retrieved easily, the UML diagram used must be described as a correspondingly directed graph. After that, all executable paths can be generated by retrieving this graph. And such a graph is named as UML Graph (UG).

In the case given in section IV below, an UG was created from UML state diagram. In this UG, a node represents a system state and a directed edge a transfer between two adjoining states.

Definition 5 Executable Path (EP). An EP is a path with one unique start node and one tail node in UG.

In some cases, UG may involve loop. Generally, a loop appears in an executable path only zero and 1 time in the path coverage of software testing. So, before the graph-retrieved algorithm is implemented, this should be configured as a constraint condition.

Note that: the graph-retrieved algorithm adopted in this paper is general and common to that we study in the course of data structure.

In this paper, a set named as PATH, {p1, p2, p3, p4…}, is defined to store all executable paths generated and each pi in it corresponds to an executable path. A detailed case will be given in section V below.

B. PITCs Generation

After the set PATH including all executable paths of SUT is generated, PITCs can be generated from it. In order to complete this process, the contents of each node and edge of a path pi in PATH should be determined and given. And the following table II is defined to do it for providing the information needed.

In such table, the state with input and output corresponding to each node is given clear. The table should be created in advance and the description of each item in it should be given accurately. The table is named as SET (state and event table).

TABLE II

THE STATE AND EVENT TABLE (SET)

State NO.	Current state	Input	Expected output

In table II above, the column State NO. corresponds to the NO. of each node in UG. The column Current state means the system state followed by the test case execution. The column Input represents the data that may be entered in the current state and Expected output the ones appeared in next system state. And the Input and Expected output together determine a corresponding data object. Exactly, each pair of the input and output required in a state of SUT is given manually by analyzing and determining the input boundary of each attribute of one data object. All input and excepted output are in the form of platform- independent data.

The following process is given to implement PITCs generation from the PATH according to the table SET created in advance. And each executable path included in PATH is processed one by one.

Step1: take the ith path pi from PATH, and then determine the initial state of the first node of pi. The initial state is the content of the column Current state in the table SET.

Step2: determine the <inj, eoutj>, the input and expected output of the jth state node in the current path pi. Here, the inj is the content of the column input corresponding to the jth state in SET, and it may be null; the eoutj is the content of the column Expected output corresponding to the jth state in SET.

Step3: continue to take the next adjoining node in the current path pi and then go to the Step2 until all nodes in pi have been processed eventually. Note that, the last node in each path pi is the end node of UG and that is marked with “END”.

Step4: according to the processed order of each executable path in PATH, each PITC generated is numbered with a string, for example tc1, tc2….

The above process from step1 to step4 will be repeated until all executable paths in PATH have completely been transformed at last.

A PITC generated according to the above process is not an executed test case because in it all attribute parameters involved are not assigned to concrete values that conform to a high programming language or platform specification. So, it must be transformed into the corresponding PSTC, one type of executable test case defined in this paper.

C. *PSTCs Generation*

In this section, the work is just to identify all data objects and their input variables involved in each PITC and then choose appropriate values from the input space for all variables.

The value space of an attribute variable generally consists of a valid subspace, in which values are expected to be accepted by SUT, and a failure one, in which values are invalid and expected to cause the SUT to produce some kind of failure response.

To implement this process, the table PISDMT (platform independent-specific data mapping table) is defined in the form of the table III below which conforms to the figure 2 and Definition 5 given in section II. Exactly, the PISDMT is used to describe the information about PIDO and PSDO and the mapping relation between them, which is very essential for the generating process from PITCs to PSTCs.

In Table III below, the column DO is used to identify each unique data object. The BSF (basic state feature) is to describe the state features, valid or invalid, of the value space of the current DO. The PSA refers to the platform specific counterpart of the current DO. The PSS (platform specification) is to describe the valid or invalid values assigned to the current DO under the final application platform.

TABLE III

PLATFORM INDEPENDENT-SPECIFIC DATA MAPPING TABLE

(PISDMT)

DO	BSF	PSA	Value Space	PSS

According to the contents of PISDMT, the detailed transformation process is defined as follows:

PITC ⓣ PSTC ≡ PIDO: (PIA, PIS) ⓣ PSDO:(PSA, PSS)

In it, the “≡” represents “being defined”. It means that the transformation process from a PITC to the corresponding PSTC(s) is equal to that from the PIAs and PISs of each PIDO in a PITC to the PSAs and PSSs of the corresponding PSDO in table PISDMT. In fact, it completes the process in which all variables in PITC are assigned to the concrete values that comply with the syntax of the final application platform determined.

The main contents of a test case usually include the initial state and a set of data pair including input and expected output. For each part of it, one corresponding transformation rule is described as follows:

(1) Transforming the initial state in a PITC into the one in a PSTC

Rule1: the initial state transformation

IF (∀PITC (∃do∈PITC.InitState Λ do==PISDMT.DO Λ ∃pss∈ PISDMT.PSS)) THEN

PITC.InitState.do← pss

Here, the “←” represents “being replaced” and the “∈” “being included”. It means that if a data object do exists in the InitState of a PITC, the do is to be replaced by the corresponding data pss included in the current row of the table PISDMT.

(2) Transforming the input and output in a PITC into the ones in a PSTC.

Rule2: the input and output data transformation

IF (∀PITC (∃do ∊ PITC.Data ∧ do==PISDMT.DO ∧ ∃pss∊ PISDMT.PSS)) THEN

PITC.Data.do←pss

It means that if a data object do exists in the <ini, outi> of a PITC, the do is to be replaced by the corresponding data pss included in current row of the table PISDMT.

Note that: two points are very important for the PITC-to-PSTC transformation process and should be further elaborated as follows.

(1) Correctness of TRs (transformation rules). TRs in the form of the above table-transformation process should keep consistent with the mapping relation showed in figure 2. And they implement the PIA-to-PSA transformation process for each data object. Because this process only assigns platform-specific values to the attributes of a data object but not changes the semantic of them, it holds the property preservation. That is, the semantic of an object in PIM/PITC cannot be changed and continues to be preserved in the corresponding PSM/PSTC.

(2) Traceability of transformation process. According to figure 2 and table III above, between the platform -independent data and the corresponding platform -specific ones is one-to-many relation. And that is also the relation between a PITC and the corresponding PSTC (s). Therefore, the PITC can be traced uniquely from a PSTC.

According to the transformation rules defined above, each PITC can be processed to be transformed into some PSTCs as follows:

Step1: According to Rule1, take a PITC tc_i from the set PITCs, then replace the DO in PITC**.**InitState with the corresponding *pss* in PISDMT**.**PSS.

Step2: Take the first data pair $<in_1,eout_1>$ of tc_i, then replace the DO in $<in_1,eout_1>$ with the corresponding *pss* in PISDMT**.**PSS. If the DO has many values in PISDMT**.**PSS, respectively generate a correspondingly new test case by using each *pss* to replace the DO until the PISDMT**.**PSS becomes empty.

Step3: Go on to process next $<in_j, eout_j>$ of tc_i by replacing all DOs in it. And repeat Step2 until all input-output data pair has completely been processed at last.

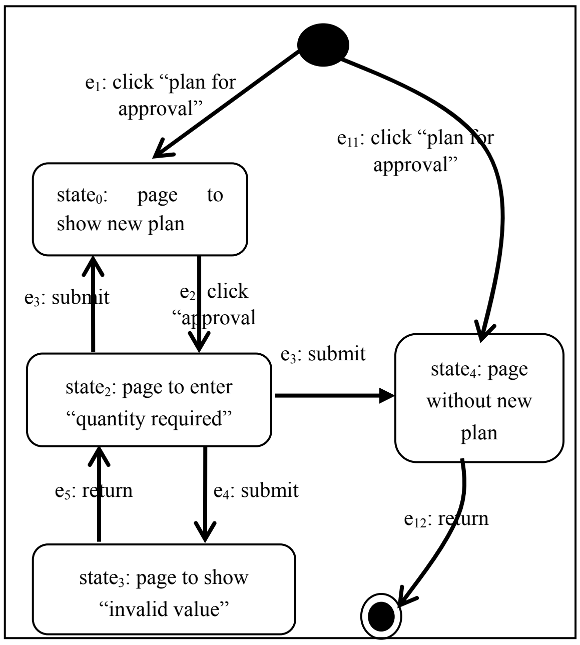
Step4: Go to Step1 to take and process next PITC tc_{i+1} of the set PITCs, and repeat from Step1 to Step3 until the set PITCs becomes empty.

In the above process, the main work is to replace each DO in each PITC with all valid and invalid data, the corresponding value pss in PISDMT.PSS. After this process, the set PSTCs can be generated and test cases in it can be executed manually.

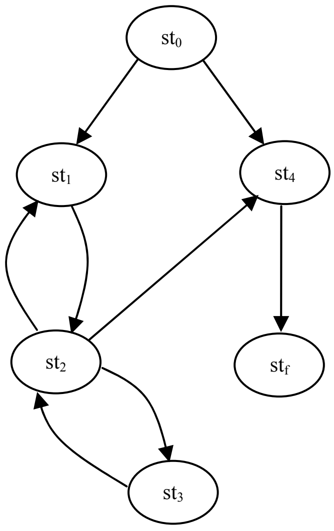
IV. A Case Study: State Diagram-Based Unit Test Case Generation

The following case is about a subsystem "power plan for approval" which can be used to online submit the power quantity for next month to the administration and apply for approval.

Step1: Create the directed graph for retrieval from the state diagram of the subsystem "power plan for approval", seen in figure 3 (b) below. Note that: each event in figure 3(a) is to be viewed as one part of the input of one source state node corresponding to it. In this abbreviated graph, the st0 corresponds to the start node in state diagram and the stf the unique end node.


e_1: click "plan for approval"
e_{11}: click "plan for approval"
$state_0$: page to show new plan
e_3: submit
e_2: click "approval
e_3: submit
$state_4$: page without new plan
$state_2$: page to enter "quantity required"
e_5: return
e_4: submit
e_{12}: return
$state_3$: page to show "invalid value"

(a) State diagram of the subsystem "power plan for approval"



(b) The corresponding graph for retrieval

Figure 3 A case: state diagram and the corresponding directed graph from it

Step2: Retrieve the graph in figure 3(b) to generate all executable paths, a set PATH, of the subsystem. The set PATH is *{p1, p2, p3, p4}*, where:

p1= (st_0, st_1, st_2, st_1, st_2, st_4, st_f) // Check and approve two plans continually, that is, including the loop path one time between st1 and st2.

p2= (st_0, st_1, st_2, st_4, st_f) // no loop between st_1 and st_2.

p3= (st_0, st_1, st_2, st_3, st_2, st_4, st_f) // loop one time between st_2 and st_3.

p_4= (st_0, st_4, st_f) // no new plan for approval.

Step3: define the state and event table (SET) as table IV below, and then, according to it and the PATH generated in step2 above, implement the process given in section III(B) to generate the corresponding set PITCs.

The PITCs generated for this case is the set {tc_1, tc_2, tc_3, tc_4}, where

(tc_1, Initial system page, {< Click "check and approve", page to show new plans >, < Click "approval", page to enter "quantity required">, <enter Valid quantity and click "submit", page to show new plans >, < Click "approval", page to enter "quantity required">, <enter Valid quantity and click "submit", page without new plans >, < Click "Return", Initial system page >});

(tc_2, Initial system page, {< Click "check and approve", page to show new plans >, < Click "approval", page to enter "quantity required">, <enter Valid quantity and click "submit", page without new plans >, < Click "Return", Initial system page >});

(tc3, Initial system page, {< Click "check and approve", page to show new plans >, < Click "approval", page to enter "quantity required">, <enter Invalid quantity and click "submit", Page to show "Invalid value">, < Click "approval", page to enter "quantity required">, <enter Valid quantity and click "submit", page without new plans >, < Click "Return", Initial system page >}); // given that the maximum is 1000, so 1001 is an invalid number.

TABLE IV

THE STATE AND EVENT TABLE (SET)

State NO.	Current state	Input	Expected output
st_0	Initial system page	Click "check and approve"	page to show new plans
st_0	Initial system page	Click "check and approve"	page without new plan
st_1	page to show new plans	Click "approval"	page to enter "quantity required"
st_2	page to enter "quantity required"	Valid quantity, click"submit"	page to show new plans
st_2	page to enter "quantity required"	Valid quantity, click"submit"	page without new plans
st_2	page to enter "quantity required"	Invalid quantity, click"submit"	Page to show "Invalid value"
st_3	Page to show "Invalid value"	Click "Return"	page to enter "quantity required"
st_4	page without new plan	Click "Return"	Initial system page

(tc_4, Initial system page, {< Click "check and approve", page without new plans >, < Click "Return", Initial system page >}).

Step4: define the table PISDMT. According to definition 4 given in section II, a data object is one with attributes that determine the input space of SUT. The table PISDMT for the subsystem "power plan for approval" can refer to Table V defined below.

TABLE V

PISDMT FOR THE SUBSYSTEM "POWER PLAN FOR APPROVAL"

DO	BS	PSA	Values space	PSS
quantity	valid	Valid quantity	[0,1000]	50/150/1000
quantity	invalid	Invalid quantity	$(1000,+\infty)/(-\infty,0)$	1001/-1/sgh123

Step5: on the basis of PISDMT, generate the set PSTCs from the set PITCs according to the process given in the subsection C of the former section III.

Note that: in this paper, the concrete values assigned to one corresponding attribute variable of a data object can be manually defined in advance after the table PISDMT is created. Of course, they can also be generated temporarily according to the value space but this can cause some performance problems such as time consuming. Generally, all valid values for data objects can be included a test case named as a success one. And each invalid value outside of the input space should individually correspond to a test case called as a failure one. So, the following set PSTCs generated in this case includes 6 executed test cases, and that is a set {tc_1, tc_2, tc_3, tc_4, tc_5, tc_6 }, where

(tc_1, Initial system page, {< Click "check and approve", page to show new plans >, < Click "approval", page to enter "quantity required">, <enter "50" and click "submit", page to show new plans >, < Click "approval", page to enter "quantity required">, <enter "150" and click "submit", page without new plans >, < Click "Return", Initial system page >}); //continue to check and approve two plans

(tc_2, Initial system page, {< Click "check and approve", page to show new plans >, < Click "approval", page to enter "quantity required">, <enter "50" and click "submit", page without new plans >, < Click "Return", Initial system page >}); //only check and approve two plans

(tc_3, Initial system page, {< Click "check and approve", page to show new plans >, < Click "approval", page to enter "quantity required">, <enter "1001" and click "submit", Page to show "Invalid value">, < Click "approval", page to enter "quantity required">, <enter "150" and click "submit", page without new plans >, < Click "Return", Initial system page >}); // given that the maximum is 1000, so 1001 is an invalid number.

(tc_4, Initial system page, {< Click "check and approve", page to show new plans >, < Click "approval", page to enter "quantity required">, <enter "-1" and click "submit", Page to show "Invalid value">, < Click "approval", page to enter "quantity required">, <enter "1000" and click "submit", page without new plans >, < Click "Return", Initial system page >}); // given that the maximum is 0, so -1 is an invalid number.

(tc_5, Initial system page, {< Click "check and approve", page to show new plans >, < Click "approval", page to enter "quantity required">, <enter "sgh123" and click "submit", Page to show "Invalid value">, < Click "approval", page to enter "quantity required">, <enter "50" and click "submit", page without new plans >, < Click "Return", Initial system page >}); // given that the value is only a number between 0 and 1000, so a string including letter is invalid.

(tc_6, Initial system page, {< Click "check and approve", page without new plans >, < Click "Return", Initial system page >}). // have no plan for approval

V. Experiments And Analysis

A.Z. Javed etc. [4] proposed an approach to model-driven component testing. According to it, the meta-models corresponding to PIM and PSM and the transformation rules from PIM to PSM ware defined respectively, and then the process to generate test cases was implemented on the basis of them. But in [4], only an idea was given and the detailed implementing method was absent. Moreover, it "generates" test cases through PIM and PSM and the transformation rules between them. This was also different from the method proposed in this paper, in which test cases were generated by the means of transformation from PITC to PSTC.

Another approach to implementing model-driven testing was to create test models corresponding to PIM and PSM respectively and then to define the transformation rules between elements of them. This way is also adopted by most researchers [2] [5]-[11]. UML 2.0 test profile had also been released as a specification by OMG in 2004. Being different from these methods proposed, the one in this paper was to generate test cases through creating meta-models of platform-independent data and platform-specific one and the mapping relationship between them. The mapping and transformation rules in it were defined in the form of relation table, which made the process of test case generation easy to be understood and implemented by users. For instance, all data involved it can easily be handled by database or other forms.

Tcases [12] is an open-source tool for black-box test case generation. With Tcases, users can define the input space for the SUT and the level of coverage that they want. Then Tcases can generate a minimal set of test cases that meets testing requirements. Tcases was guided by the coverage of the input space of SUT. In Tcases, the input space and the functions of SUT were defined and described in two individual XML files respectively. It used input values to generate two types of test cases — "success" cases, which use only valid values for all variables, and "failure" cases, which use a failure value for exactly one variable.

PItoPSTcases is a simple tool to generate test cases, which implements the method proposed in this paper. It was developed by our team using Eclipse (SDK 3.7). In it, all related tables were described in the form of databases of SQL Server. The architecture of PItoPSTcases is illustrated in the figure 4 below.

The main differences between two tools are listed in the table VI below. According to the content of it, the input space and functions of SUT must be created in the form of XML file respectively before Tcases runs. And the functions here are used to describe the logic of SUT.

But, for PItoPSTcases, only data object and the mapping relation between PIDO and PSDO are to be described as tables, and the logic of SUT can directly be obtained from the selected UML design model and the graph form it. So, on the basis of the table SET and PISDMT created in advance, it can generate the executed test cases, PSTCs. The values of an attribute variable were selected and configured manually in the form of table. To some extent, this improves the accurateness of test cases generated. Additionally, the section of input data is also easier to be implemented than that in Tcases. However, the accurateness of test cases generated by using Tcases are heavily dependent on that of XML files, in which all variables and the conditions or constraints related to them must be recorded accurately.

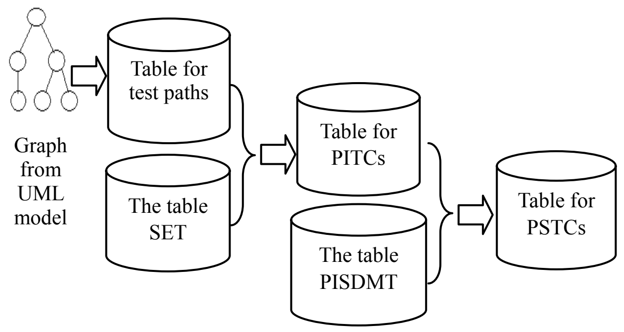


Figure 4 The architecture of PItoPSTcases

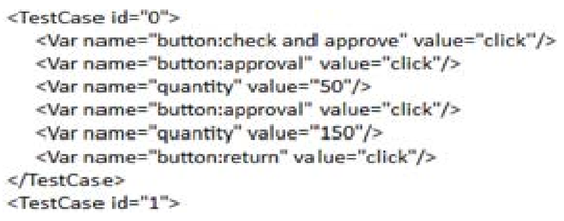
TABLE VI
DIFFERENCES BETWEEN TCASES AND PITOPSTCASES

Comparison item	Tcases	PItoPSTcases
model type	XML file	UML/graph from UML
Input	(1)Xml for input space of SUT (2)Xml for functions of SUT	(1)Table for describing data objects (2)Table for the mapping from PITC to PSTC
Output	Xml for test cases	Table(Text string) for test cases

Other tools given in some researches were to generate test cases just by selecting data from the input and output space in a random way. This can also make users face the problem that the selected data cannot satisfy the testing requirements well.

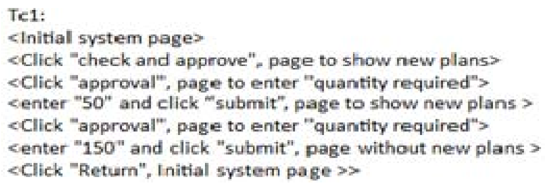
Both Tcases and PItoPSTcases were implemented at the same environments as follows: (1) OS: windows 7 core 64; (2) hardware: 10 computers with CPU (Intel(R) Core (TM) i5-3320M, 2.60GHz) and RAM 8GB.

Two fragments of screen shot of their output are respectively given in the following figure 5 and figure 6.



```
<TestCase id="0">
    <Var name="button:check and approve" value="click"/>
    <Var name="button:approval" value="click"/>
    <Var name="quantity" value="50"/>
    <Var name="button:approval" value="click"/>
    <Var name="quantity" value="150"/>
    <Var name="button:return" value="click"/>
</TestCase>
<TestCase id="1">
```

Figure 5 Output format for Tcases



```
Tc1:
<Initial system page>
<Click "check and approve", page to show new plans>
<Click "approval", page to enter "quantity required">
<enter "50" and click "submit", page to show new plans >
<Click "approval", page to enter "quantity required">
<enter "150" and click "submit", page without new plans >
<Click "Return", Initial system page >>
```

Figure 6 Output format for PItoPSTcases

The following figure 7 showed that two tools, Tcases and PItoPSTcases, were respectively compared, from two perspectives, the average time (run 10 times) spent to create the input and the one spent to generate test cases for the same subsystem "power plan for approval".

From the figure 7 below, it can be concluded that the average time spent in preparing the input by PItoPSTcases was lower about 37.5% than that spent by Tcases. And the average time spent by PItoPSTcases to generate test cases is lower about 43% than that spent by Tcases.

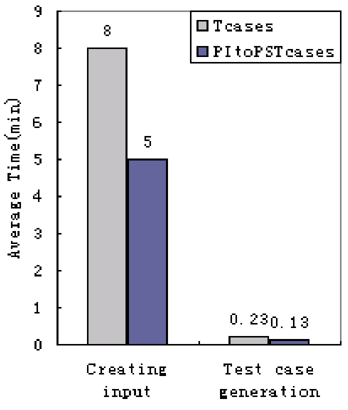


Figure 7 The average time comparison: that of creating input and that of generating test cases

VI. CONLUSIONS

To design and generate test cases is one of the most important steps to implement software testing. And Based on the idea of MDD, one method was proposed in this paper to generate executed test cases. All input data are described in the form of table which can be created and used easily. And a simple experiment given in section V showed that the method had a larger advantage of efficiency in time spent.

Of course, the type of test case defined in this paper is only executed by hand now. The next work for us is to improve the tool to generate test cases in the form of script which can directly be executed by some test tools such as xUNIT.

REFERENCES

[1] OMG. *Ptc/04-04-02: UML 2.0 Testing Profile*, Finalized Specification.
[2] R. Zhen. Model-Driven Testing with UML 2.0 [C]. In: Proceeding of Second European Workshop on Model

Driven Architecture (MDA), Canterbury, Kent, University of Kent (2004), pp. 179-187.

[3] S. Anand, E. Burke and T. Chen *et al*. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*. 2013, Vol. 86, issue 8, pp. 1978-2001.

[4] A. Javed, P. Strooper and G. Watson. Automated Generation of Test Cases Using Model-Driven Architecture[C]. In: Proceeding of Second International Workshop on Automation of Software Test, 2007. AST '07. pp. 1-7.

[5] Q. Yuan, J. Wu, C. Liu and L. Zhang, A model driven approach toward business process test case generation[C]. In Proc. of the 10th International Symposium on Web Site Evolution (WSE), 2008, pp. 41-44.

[6] M. Mussa, S. Ouchani, W. Sammane and A. Hamou-Lhadj. A Survey of Model-Driven Testing Techniques [C]. In: Proceeding of 9th International Conference on Quality Software, 2009, QSIC '09. pp. 167-172.

[7] N. Li, Q. Ma and J. Wu *et al*. A Framework of Model-Driven Web Application Testing[C]. In: Proceeding of 30th Annual International Computer Software and Applications Conference, 2006. COMPSAC '06. Vol. 2, pp. 157-162.

[8] J. Gutierrez, M. Escalona and M. Mejias *et al*. An approach for Model-Driven test generation[C]. In: Proceeding of Third International Conference on Research Challenges in Information Science, 2009. RCIS 2009. pp. 303-312.

[9] D. Mathaikutty, S. Ahuja, A. Dingankar and S. Shukla. Model-driven test generation for system level validation[C]. In: Proceeding of IEEE International High Level Design Validation and Test Workshop, 2007. HLVDT 2007.pp. 83-90.

[10] F. Wang, S. Wang, and Y. Ji. An Automatic Generation Method of Executable Test Case Using Model-Driven Architecture[C]. In: Proceeding of 2009 Fourth International Conference on Innovative Computing, Information and Control (ICICIC), 2009, pp. 389-393.

[11] M. Felderer, P. Zech and F. Fiedler *et al*. Model–driven System Testing of Service Oriented Systems [C]. In: Proceeding of the 9th International Conference on Quality Software (QSIC'2009), 2009. pp. 1-8.

[12] tcases - A model-driven test case generator. http://code.google.com/p/tcases/ (May, 2013)

D. Wei Now he is a PHD candidate of college of computer science & technology in Shandong University. His interests are software development and testing.

L. Tang He is a PHD of Inspur Genersoft Co., Ltd. His interests are ERP software development &testing.

X. Li He is a Professor of college of computer science & technology of Shandong University and also the corresponding author of this paper. His interests are software development & testing.

L. Shang He is a PhD of University of Science and Technology of Lille, Lille, France. His interests are distributed computing and software quality.