Testing Approaches for an Electricity Market Simulator

Anca Ioana ANDREESCU, Ana Ramona BOLOGA
The Bucharest University of Economic Studies, Romania
Anca.andreescu@ie.ase.ro, Ramona.bologa@ie.ase.ro

Software testing methodologies represent different approaches and techniques to ensure that a particular software application is fully tested. This paper addresses the functional testing of a software system for simulating electricity markets. The tested functionalities are briefly presented and the main testing techniques compatible with this type of system are analyzed. The purpose is to recommend the best combination of techniques recommended in this case and to present results obtained by applying them to the simulator.
Keywords: *functional testing technics, behavioral testing, electricity market, simulator*

1 Introduction

Software testing is one of the main stages of the development cycle of an IT system, vital for software quality assurance. According to IEEE, software testing can be defined as: "the process of exercising or evaluating a system or system component by manual or automatic means to verify that it satisfies the requirements" [1]. Often, however, software testing involves considerable costs and time, hence the continuous concern for reducing costs and streamlining the testing activity.

The test cases can be generated based on the results obtained in the analysis and design stages of the computer system or they can be obtained based on the code developed in the implementation stage.

This work will focus on functional testing and the main techniques recommended to perform a more complete and efficient testing of a computer system developed to simulate participation in electricity markets.

The paper is structured in four parts, as follows: section 2 - a brief presentation of software testing approaches for complex systems; section 3 - the main functionalities a wholesale electricity market simulator for which test cases have to be prepared; section 4 – presents examples of various testing techniques that may be applied to the simulator, including their advantages and disadvantages; section 4 is dedicated to results analysis, discussions and conclusions.

2 Testing in complex software systems

Software testing methodologies represent the different approaches and ways to ensure that a particular software application is fully tested. Software testing methodologies must include a broad spectrum of elements, from testing individual module units, testing the integration of an entire system, to specialized forms of testing such as security and performance [2]. According to the types of requirements of a computer system, two general levels can be identified at the level of testing: functional testing and non-functional testing, each having a series of components presented in Fig.1.

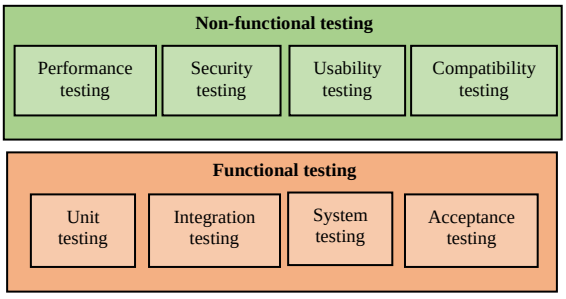


Fig. 1. Types of testing for software systems [2]

Functional testing (black box) is performed using the functional specifications provided by the client or by using the analysis and design specifications, such as use cases.

Non-functional testing (white box) involves testing the system in accordance with non-functional requirements, which usually involves measuring / testing the system

according to the defined technical qualities, for example vulnerability, scalability or usability. Traditionally, software testing considers only static views of the code and does not sufficiently address the dynamic behavior of the computer system. But, during this stage, the testing of an entire system should be performed, based on its specifications, considering the specific dynamics and interdependencies.

If in the first case we are talking about *structural testing*, based on the code resulting in the coding stage, in the second case we are talking about *behavioral testing* based on the requirements and design specifications. The use of the models obtained during the analysis and design stages can lead to a reduction of the costs through reuse and the improvement of the verification and validation. In addition, the test activity can be started in parallel with the writing of the code, as soon as the modeling results are available.

For example, in the context of object-oriented modeling using UML (Unified Modeling Language), use cases and corresponding sequence and collaboration diagrams, class diagrams, can be used as sources of information relevant to the test cases. The analysis of the use case scenarios offers a complete understanding of the system, but, being an informal description, it is difficult to generate automatic test cases. Therefore, an active research direction is the formalization of specifications to make it possible to automatically generate test cases.

Ryser proposed a method of creating test cases starting from use cases, scenarios and use case diagrams called SCENT approach [3]. Use cases and scenarios are transformed into semi-formal representation state charts that are further used for test case generation. But the sequence dependencies between use cases was first approached by Briand and Labiche [4], who used an activity diagram of use cases for each actor in the system. Later, Touseef and Zahid [5] used the same approach for representing sequential dependencies between use cases, but added some execution contracts for use case scenarios in form of logical expressions.

Use cases and sequence diagrams were used by Swain et al. [6] to generate test cases, taking into consideration the dependency sequences between use cases in form of a graph.

3 Main functionalities of the Wholesale Electricity Market Simulator

The system for simulating the participation of a producer / supplier / trader / consumer of electricity in different types of markets aims to identify how decisions regarding the price or quantities offered can influence profit optimization. It is a complex system, with numerous restrictions and operating rules, but also with important interdependencies of the decisions made on the different markets.

In this section we will describe the simulator's functionalities regarding the transactions on the following markets ([7], [8], [9], [10] and [11]): Bilateral Wholesale Markets (BWM), Day Ahead Market (DAM), Intra-Day Market (IDM), Balancing Market (BM) and Ancillary Services Market (ASM). Also, before simulating the market trading, several basic configurations are needed to be made in a General Settings Module.

For the implementation of the simulator, a series of classes have been identified that contain the entities participating in its construction. The identified classes represent abstractions of the objects that interact within the system. A distinction can be made between two classes of classes: basic classes, which include objects characterized by dynamic behaviour and independent classes, which have been used to retrieve data from other external systems. The class diagram in Figure 2 includes the independent classes found at the simulator level.

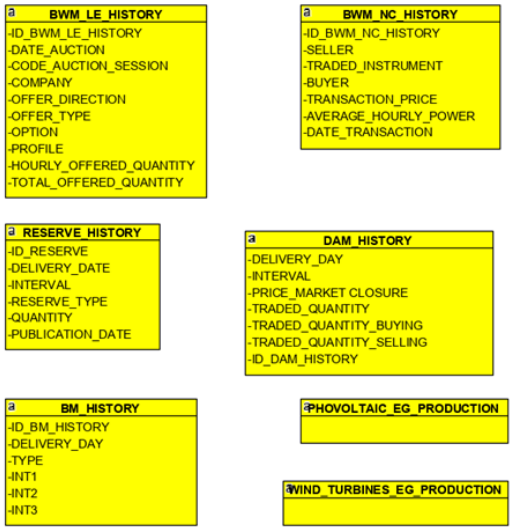


Fig. 2. Class diagram for the identified independent classes

A brief description of the information included in the independent classes is presented below:

• BWM_LE_HISTORY- Average hourly prices for deficit and surplus for the Balancing Market, contract type LE;

• BWM _NC_HISTORY - Average hourly prices for deficit and surplus for the Balancing Market, NC contract type (non-compliant);

• DAM_HISTORY - Historical energy hourly prices for DAM;

• RESERVE_HISTORY - Historical hourly prices for ancillary services;

• BM_HISTORY - Average hourly prices for deficit and surplus for the Balancing Market;

• PHOTOVOLTAIC_EG_PRODUCTION - Data for estimating photovoltaic energy production;

• WIND_TURBINES_EG_PRODUCTION Data for estimating wind energy production.

In order to model the dynamic aspects of the trading simulator, UML interaction diagrams were constructed. These diagrams are made up of a set of objects and the relationships between them, including messages that the objects send from one to the other. There are two types of interaction diagrams: the sequence diagram and the communication diagram. The two diagrams are semantically equivalent and can be transformed from one another. The sequence diagrams illustrate the messages exchanged between the actors / users of the system and the computer system, through the interfaces that it offers. We have started with simple versions of the message sequences, and later we detailed the involved objects.

Figure 3 shows the detailed sequence diagram for configuring the basic options in the General Settings Module. This was built by detailing the messages exchanged between the objects participating in the creation of the scenario. The scenario that underlies this sequence diagram is the following: the user chooses one of the predefined options to select which type of business is simulated: producer, supplier, trader or consumer. If it is a producer, it will choose whether it is type or controllable (thermo, hydro, nuclear) or uncontrollable (photovoltaic, wind). Subsequently, the user will enter the volume of electricity available for trading. The total available power per hour will be introduced, meaning the installed power, if it is a producer or the power required for trading or buying if it is a supplier, consumer or trader. Then, a technological minimum must be entered. The user must choose from a predefined list the periods for the simulation. Then, the month of the year with which the simulation begins will be chosen. The options will be saved, and the offer will be created.

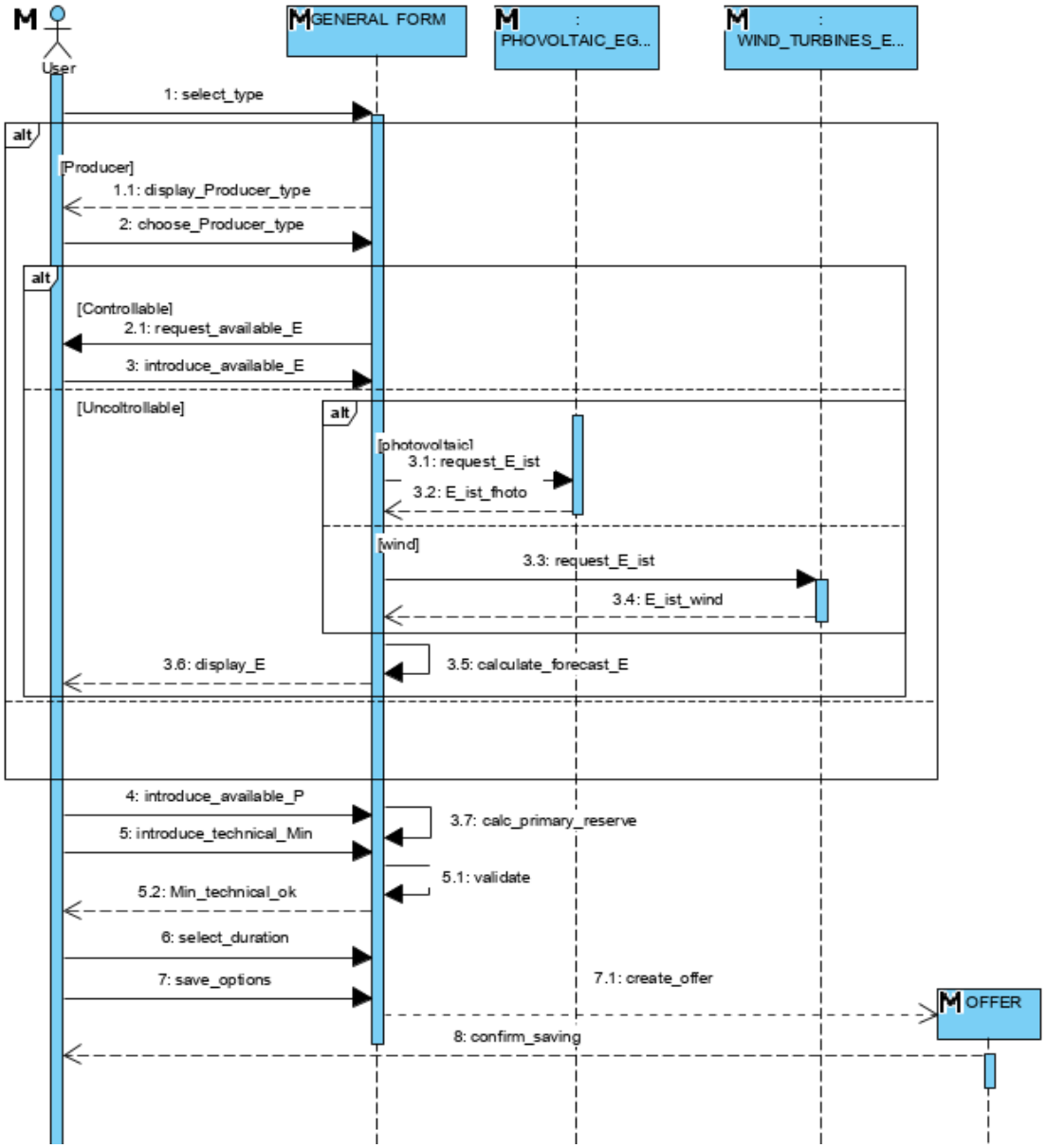


Fig. 3. Detailed sequence diagram for General Settings configurations

The sequence diagram in the figure 4 describe the simulation of trading on BWM and it is based on the following scenario: the user selects one of the options for sale or purchase on the BWM. Then, the user chooses from a predefined list of standard products offered by the Romanian Gas and Electricity Market Operator (OPCOM) the one for which the simulation of the transaction is desired, in accordance with the time-related settings in the general module. Once the selection has been made, a validation will be performed and will block the products in the list that have overlaps with the selected product. Subsequently, the user selects information specific to the chosen product, such as semester, quarter, month or year. Will be available for access only the products that have an equal or a shorter period than the chosen period for simulation in the general module. Then, the user enters the desired hourly power to be contracted. This is validated in accordance with the settings in the general module or with other transactions performed on other markets. The cost of imbalances must be introduced, as a percentage of revenues, implicitly with

the value of 10%. This default percentage can be changed. The total hourly power offered on the BWM is calculated and displayed for the chosen option (sale / purchase). Data is saved.

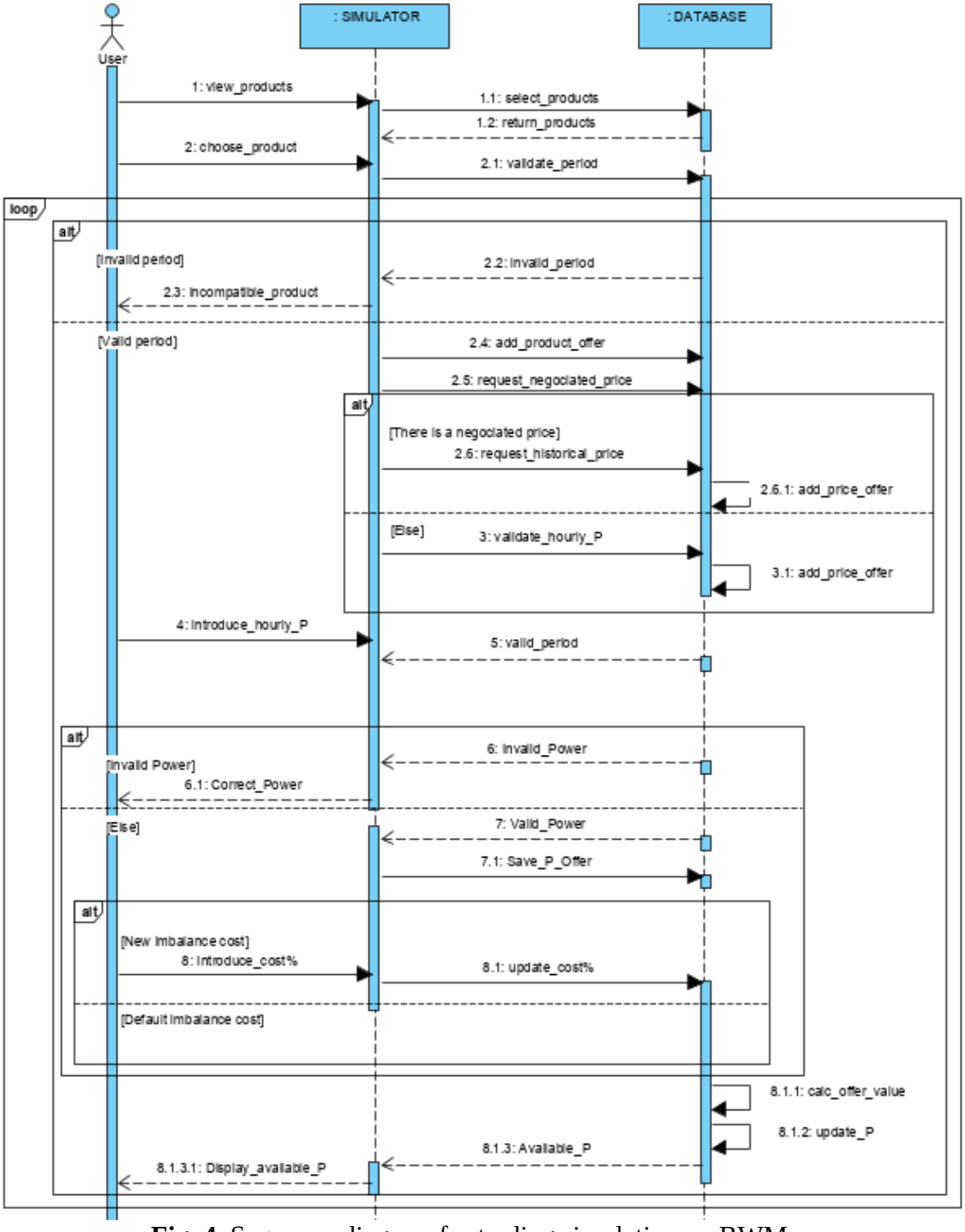


Fig. 4. Sequence diagram for trading simulation on BWM

DAM transactions can be simulated using the following scenario: the user selects one of the options for sale or purchase on the DAM and the desired month for simulation. The number of months for which the simulation is performed must be less than the period specified in the General Settings Module and correspond to the periods of a possible simulation performed on the BWM. The available hourly power is displayed by correlating the settings in the general module with any sales

on the BWM and with the primary regulation reserve (for Producers). For information purposes, a historical monthly average price for the chosen month is displayed. Also, a correction coefficient for the selected month will be displayed. It has a default value of 25% and can be modified by the user. Average monthly per hour prices will be displayed for the period chosen by the user. These can be modified. The hourly power that is to be traded must be introduced. For selling, it must be checked that the maximum available power is not exceeded. The data is saved, and the total offer is displayed.

Trading simulation on ASM is based on the process described in the following. The producer selects the desired month for simulation. It is considered that the number of months for which the simulation is performed must be less than the period specified in the General Settings Module and correspond to the periods of a possible simulation performed on BWM or DAM. For this scenario, a historical price is displayed and used. It can be modified by the producer performing the simulation. The available power for reserves is completed and validated based on the total power offered in the general settings module and of the power contracted on BWM and DAM. If desired, the correction coefficient can be updated. Following data is saved. The simulator will calculate and display the total offer on ASM.

The next scenario was considered for trading simulation on BM: the producer selects the desired month for simulation. The number of months for which the simulation is performed must be less than the period specified in the General Settings Module and correspond to the periods of a possible simulation performed on the BWM or DAM. Historical monthly average prices per hour for deficit and surplus will be displayed. Historical prices can be modified by the user. The power offered by the producer on this market is obtained as the difference between the total power completed in the general settings module and the quantities contracted by participating on other types of markets. Automatically a correction percentage is applied to the estimated revenues or expenses for the simulated period. By default, the risk level is 30% and can be changed. Data for the current month and the offer are saved. Total updated offer is displayed.

4 Recommended testing techniques

The main components underlying the testing of computer systems are the techniques, activities, instruments and the controlled environment. Of the mentioned components, we will focus on the techniques used in the testing process. The techniques of designing the test cases can be divided into three categories [12]:

a) Test techniques based on specifications or black box;

b) Testing techniques based on structure or white box;

c) Testing techniques based on experience.

In the following, several types of specification-based techniques (black box) are presented in detail. These tests derive from specifying the desired behavior of the system. It starts from the basic idea that specifications should define what a system should do, not how behavioral specifications should be implemented.

4.1 Partitioning into equivalence classes

The documents containing the system requirements normally indicate the rules that the system must follow. There may be rules that imply belonging to a certain range of values or there may be rules of the type if ... then. For example, Rule A might be "if n is less than one, then this is executed." Rule B could be "if n is greater than or equal to one and less than or equal to twelve, execute this".

We consider the rules for the values that a field can store for one month of the year. Thus, we have the following situations:

a) if the month is less than one, the value is not valid;
b) if the month is between one and twelve, it is admissible and the value can be accepted;
c) if the month is greater than twelve, the value is invalid and an error is displayed.
The entire infinite range of integers that could be introduced into the system for the value of the month must fit into one of these three criteria, one of these categories or classes: a) smaller than one; b) between one and twelve; c) greater than twelve. If a value is selected from each number class and we use it as a test case, it can be said that all three rules have been covered. Also, we can say that each value belongs to an equivalence class, hence the name of the technique.

There are three criteria that must be met when creating equivalence classes [13]:
• *Coverage*: each possible input value must belong to one of the equivalence classes.
• *Disjoint character*: the same input value cannot belong to more than one equivalence class.
• *Representation*: If, by using as input value a particular member of an equivalence class, the execution results in an error state, then the same state can be detected using as input value any other member of the class.
Partitioning into equivalence classes applies to all types of values, whether they are continuous or discrete. Using the scenario with the months of the year presented above, the following table can be defined for the partitions of inputs and outputs:

Table 1: Partitioning into equivalence classes for inputs and outputs

Rule	**Input partition**	**Output partition**
Month less than 1, display "Invalid, value too small"	Month <1	Invalid, too small
Month between 1 and 12, display "Valid"	(Month>=1, Month <=12)	Valid
Month greater than 12, display "Invalid, value is too big"	Month >12	Invalid, too big

4.2 Analysis of limit values

The errors tend to be grouped around the limit values. The assumption underlying this type of testing is that developers often omit special cases, represented by the "borders" of equivalence classes.
Using the previous example, we can see that in order to be valid, the month will have to fall within the "valid limit values" from 1 to 12. Also, for the month to be invalid, it must be outside these valid limits. The principle of testing with the limit value analysis is to use the limit value itself and another value as close to it as possible, to be able to reach any part of the limit, using the precision that was applied to the partition. In the example presented, 0,1,2 are valid values for lower limit testing, while 11,12,13 are valid values for upper limit testing.
We can say that the limit values analysis is a particular case of the test method based on equivalence classes, which is centered on exploring the limit cases. Instead of choosing an arbitrary representative of the equivalence class for testing, the method involves choosing a "boundary" element, which represents a particular case. A good way to represent valid and invalid partitions and boundaries is in a table with the following structure:

Table 2. Example of partitions and limit values

Testing conditions	Valid partitions	Invalid partitions	Valid limits	Invalid limits
Average hourly price	10 RON –100 RON	<10 RON >100 RON	10 RON 100 RON 10,001 RON 99,999 RON	9,999 RON 100,001 RON

4.3 Testing based on decision tables

Equivalence class partitioning techniques and boundary value analysis are often applied to specific situations or inputs. However, if different combinations of inputs lead to certain actions, this may be more difficult to show using partitioning into equivalence classes and analyzing limit values, techniques that tend to be more focused on the user interface. The other two test techniques based on specifications, decision tables and transitions between states are more focused on business logic or business rules.

A decision table is an effective way of dealing with combinations of elements (for example, input data). This technique is sometimes referred to as the *cause-and-effect table*. Decision tables provide a systematic way of specifying complex business rules, which is useful for both developers and test engineers. Decision tables can be used in designing test cases, whether or not they are described in the specifications, as it helps test engineers explore the effects of different input combinations as well as other system states that need to correctly implement business rules. Decision tables help systematically select efficient test cases and can have the beneficial side effect of finding problems and ambiguities in specifications. It is a technique that works well in combination with partitioning into equivalence classes. The combination of the conditions explored can be a combination of equivalence partitions.

In using the decision tables for designing the tests, the first step is to identify a function or subsystem that has a behavior that reacts according to a combination of inputs or events. After identifying the elements to be combined, they can be placed in a table that shows all the combinations of True and False for each of the conditions. The next step is to identify the correct result for each combination. Each combination of inputs and outputs is found in the specialized literature as a rule. From a testing perspective, a rule is a test case.

The outputs can be represented either in the form of an expected result (a single output line), or by listing each type of action that can be considered a result and specifying whether or not it will be performed according to the values of the input conditions. We illustrate below the two variants for specifying the results for the decision tables. For the first case, we create test cases for the purpose of verifying a login form, having the structure in the figure below.

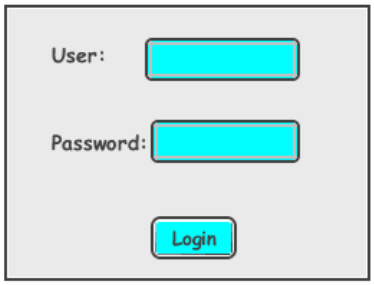


Fig. 5. Login form

The constrained decision table for testing the functionality of the login form is presented in Table 4. The entry conditions verify that the user and password data have been entered.

Table 3. Decision table for testing the login form

Input	**CT1**	**CT2**	**CT3**	**CT4**
User	Yes	Yes	No	No
Pass-word	Yes	No	Yes	No
Login	Yes	Yes	Nu	No
Expected output	System respon	Error message	Error message	Error message

The following decision table was constrained to test the functionality of the electricity market simulator, having as input conditions the type of market that is simulated trading and the type of user. The expected results model the possibility of selling / increasing or buying / decreasing in one of the markets depending on the type of user.

Table 4. Decision table for testing the trading possibilities in the market simulator

Input	**1**	**2**	**3**	**4**	**5**	**6**
Producer	Yes	No	No	No	Yes	No
Supplier	No	Yes	No	No	No	Yes
Trader	No	No	Yes	No	No	No
Consume	No	No	No	Yes	No	No
BWM	Yes	Yes	Yes	Yes	No	No
DAM	No	No	No	No	Yes	Yes
BM	No	No	No	No	No	No
ASM	No	No	No	No	No	No
Selling/ Growth	Yes	Yes	Yes	No	Yes	Yes
Buying/ Drop	Yes	Yes	Yes	Yes	Yes	Yes

4.4 Testing based on the transition between states

Testing based on the transition between states is used when a certain aspect of the system can be described in the form of what is called a "finite state machine". This shows that the system can be in a (finite) number of different states, and the transitions from one state to another are determined by the rules of the "machine". This model is based on both the system and the test cases.

Any system where a different output can be obtained for the same input, depending on what happened before, is a finite state system. Such a system can be described in the form of UML diagrams of state machines. A model that highlights the transition between states has four basic parts:

• The states in which the system can be located (for example, closed or open)

• Transitions from one state to another (transactions are allowed only between certain states)

• The events that generate the transitions

• Actions resulting from transactions.

It is noteworthy that, in any state, an event may cause a single action, while the same event, from a different state, may cause another action or the transition to another state.

The test conditions can be derived from the diagram that models the machine with states in different ways. Each state can be considered a testing condition, as is each transition. The following table describes a general model for a state table that can be used to test a system in which finite states can be identified and modeled.

Table 5. General model for a table of states

		1st test case	2nd test case	3rd test case	4th test case
		1st transition	2nd transition	3rd transition	4th transition
STATES	A state				
	B state				
	C state				
	D state				

One of the advantages of the technique based on the transitions between states is that models can be built at the desired levels of detail and abstraction to verify the different critical or less critical aspects of the system.

4.5 Use case-based testing

Use case-based testing is a technique that helps identify test cases for verifying the entire system from a transactional perspective. A use case is a description of a particular way of using the system by an actor. Each use case describes the interactions that the actor has with the system to perform a specific task (or at least a measurable result for the user). In general, actors are people, but there may be other systems. Use cases are a sequence of steps that describe the interactions between the actor and the system.

The use cases are defined in terms of the actor and from his perspective, not of the computer system. I often use business language and terms, rather than technical

terms. They serve as a basis for developing test cases for system testing and acceptance testing. Use cases may reveal integration defects, i.e. defects caused by the incorrect interaction between different components.
Each use case usually has a main scenario, as well as other additional alternatives (covering, for example, special cases or exceptional conditions). Each use case must specify any preconditions that must be met to operate the use case. The use cases must also specify the postconditions that can be observed and a description of the final state of the system, after the use case has been successfully executed.
The system requirements can be specified as a set of use cases. This approach can facilitate the involvement of users in the process of collecting and defining the requirements, but also in the testing process.The following table shows the partial variant of a use case template that documents the simulation of the sale on the BWM market for the manufacturer. The basic and alternative flows are highlighted, with the highlighting of the actors involved. These will serve as input into the process of testing the scenarios involved in this use case.

Table 6. Partial description of a use case highlighting the actors involved

Name	Simulation sale on the BWM market for the manufacturer
Preconditions	The manufacturer specific activity settings from the general module have been introduced
Postconditions	The manufacturer has simulated its offer on the market of bilateral contracts for the period specified in the general module. The simulation data on the BWM were saved.
Basic flow **P: Producer** **S: System**	1. P: Select the option to sell on the BWM market. 2. P: Choose from a predefined list of products (OPCOM) the one for which you want to simulate the transaction, according to the time settings in the general module. 3. S: Check overlaps with the selected product. 4. P: Selects information specific to the chosen product, such as semester, quarter, month or year. (Only available products that have a shorter or a shorter period with the chosen period for simulation in the general module can be accessed.) 5. P: Enter the desired hourly power to be contracted. 6. S: The hourly power desired to be contracted will be validated in order not to exceed the declarant availability in the general module. 7. S: A historical monthly average price for the selected product will be displayed 8. S: The cost of imbalances is completed, as a percentage of the receipts, implicitly with the value of 10%. 9. S: Calculate and display the total hourly power offered on BWM for sale. 10. S: Saves data for BWM simulation.
Alternative flows	3A: There are overlaps with the selected product - S: The products in the list that have overlaps with the selected product will be blocked.

6A: The hourly power exceeds the declarant available in the general module- S: An error message is displayed and the hourly power is re-entered
7A: The manufacturer wants to change the price - P: Changes the historical price
7B: There is a negotiated hourly price for a product - S: it will automatically fill in and use the negotiated price
8A: Manufacturer wants to change the cost of imbalances - P: Change percentage of imbalances

As an example of creating and using dependency charts, we selected only the participation of the energy producer on two electricity markets: BWM and DAM. The scenarios in which the Producer is the actor are:

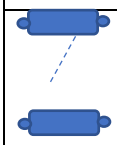
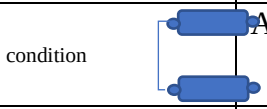
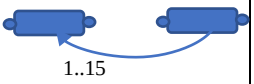
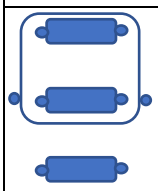
(1) Basic configurations in the General module;
(2) Select product for BWM transaction;
(3) Fill in the hourly power to be contracted for the listed products;
(4) Fill in the price of the imbalances as a percentage of the receipts;
(5) Estimate the correction coefficient according to the weather forecast;
(6) Fill in the offer for each time interval of each month.

The scenarios in which the System is the actor are:

(10) Import historical data about products;
(11) Retrieve the predefined list of products and details;
(12) Calculate and display Average historical price per hour for the selected product;
(13) Hourly power validation;
(14) Block those products from the list that have overlaps with the selected product;
(15) Estimate the total contracted quantity and update the availability;
(16) Retrieve the historical DAM data;
(17) Calculate monthly average of the historical price;
(18) Validate offer per interval and per month.
(19) Total offer amount update

Dependency charts are very useful in testing, as they support derivation of additional test cases and ensure that dependencies between scenarios are tested. Dependency chart in Figure 6 uses the notations presented in Table 7.

Table 7. The notations for dependency charts [3].

Symbol	**Explanation**
	Unrestricted scenario
	General dependency
	Sequence
	Alternative
	Iteration
	Real time dependencies
	Structuring constructs

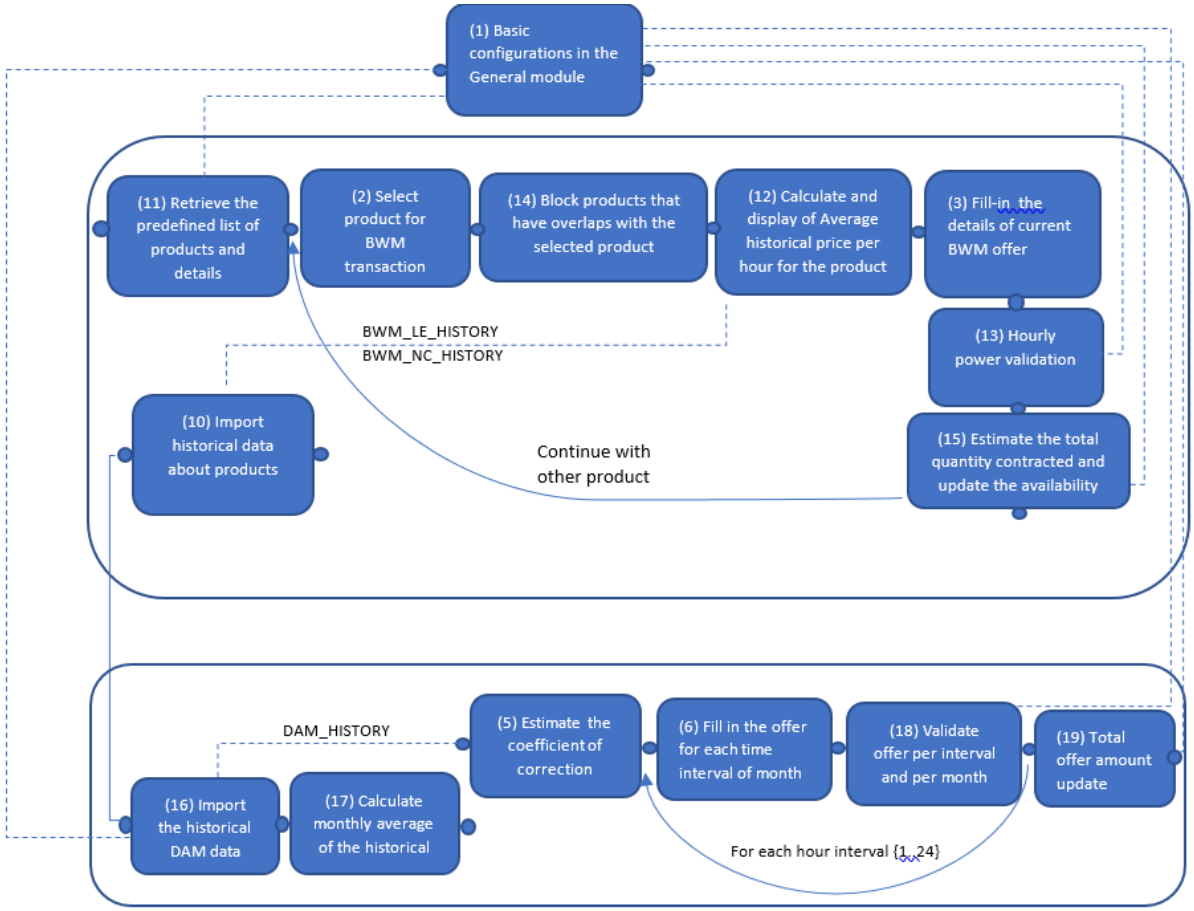


Fig. 6. A dependency chart for the electricity market simulator

Table 8. Examples of test case identification

Test preparation		Producer has entered basic configurations in the General Module	
ID	**Scenarios**	**Expected result**	**Description**
1.1	(11), (2), (14)	User can not be able to add the selected product to his offer	User select products on BWM that overlaps the previous selected product(s)
1.2	(11), (2), (14), (12), (3), (13)	An error message is displayed and the hourly power has to be re-entered	Hourly power exceeds the declarant available in the general module
1.3	(16), (17), (5), (6), (18)	An error message is displayed and per interval and per has to be re-entered	Offer per interval and per moth on DAM exceeds the declarant available in the general module

Use cases present their scenarios in the form of all possible paths for the specific functionality of the computer system. Therefore, all these scenarios must be verified for the actual implementation of the testing process. By crossing the dependency chart, the necessary test paths can be identified and the use cases efficiently generated. Table 8 presents some examples of test cases based on scenario dependencies, where scenarios are indicated using their identification numbers between brackets.

5 Conclusions and future work

As our first development phases used UML for modeling, we find best fitted UML-Based Approach to System Testing, with its obvious advantages.

Each of the functional testing techniques presented above are frequently applied, each

with its own recommendations. Often these are used in a complementary manner to benefit from their specific advantages. In the case of a complex system, such as the system for simulating the participation in the electricity markets presented in this paper, where there are numerous and interdependent restrictions between the different functionalities, the test based on use cases will be the most appropriate technique.

6 Acknowledgment

This paper presents the scientific results of the project "Intelligent system for trading on wholesale electricity market" (SMARTRADE), co-financed by the European Regional Development Fund (ERDF), through the Competitiveness Operational Programme (COP) 2014-2020, priority axis 1 – Research, technological development and innovation (RD&I) to support economic competitiveness and business development, Action 1.1.4 - Attracting high-level personnel from abroad in order to enhance the RD capacity, contract ID P_37_418, no. 62/05.09.2016, beneficiary: The Bucharest University of Economic Studies

References:

[1] N. Kosindrdecha, and J. Daengdej. *A test case generation process and technique* Journal of Software Engineering, 2010;

[2] Inflecta, *Software Testing Methodologies - Learn The Methods & Tools*, March 2018, https://www.inflectra.com/ideas/topic/testing-methodologies.aspx, last accessed on 13.12.2019;

[3] J. Ryser, and M. Glinz, *A scenario-based approach to validating and testing software systems using statecharts.*, Proc. 12th International Conference on Software and Systems Engineering and their Applications, 1999;

[4] L. Briand and Y. Labiche. *A UML-based approach to system testing*, Software Quality Engineering Laboratory, Systems and Computer Engineering, Innovations in Systems and Software Engineering, Springer. pp. 12-24, 2002;

[5] M. Touseef and Z. H. Qaisar. *A use case driven approach for system level testing*, International Journal of Computer Science Issue. Vol. 9, 2012;

[6] S.K. Swain, D. P. Mohapatra, and R. Mall, *Test case generation based on use case and sequence diagram*, International Journal of Software Engineering", 2010;

[7] Guideline on Electricity Balancing. Retrieved January 27, 2020, from https://www.entsoe.eu/network_codes/eb/ ;

[8] Guideline on Electricity Transmission System Operation. Retrieved January 27, 2020, from https://www.entsoe.eu/network_codes/sys-ops/ ;

[9] Regulation regarding intra-day market operation of Romanian Energy Regulatory Authority (RERA). Retrieved January 27, 2020, from https://www.anre.ro/ro/legislatie/documente-de-discutie-ee1/proceduri-oper-regl-comerciale/regulamentul-de-organizare-si-functionare-a-pietei-intrazilnice-de-energie-electrica1387366406 ;

[10] Regulation regarding day ahead market operation of Romanian Energy Regulatory Authority (RERA). Retrieved January 27, 2020, from https://www.anre.ro/ro/energie-electrica/legislatie/documente-de-discutie-ee/proceduri-oper-regl-comerciale/regulament-de-organizare-si-functionare-a-pietei-pentru-ziua-urmatoare-de-energie-electrica-cu-respectarea-mecanismului-de-cuplare-prin-pret-a-pietelor&page=1 ;

[11] Regulation regarding negocitated contracts of Romanian Energy Regulatory Authority (RERA). Retrieved January 27, 2020, from

https://www.anre.ro/ro/legislatie/documente-de-discutie-ee1/proceduri-oper-regl-comerciale/regulament-privind-modalitatile-de-incheiere-a-contractelor-bilaterale-de-energie-electrica-prin-licitatie-extinsa-si-negociere-continua-si-prin-contracte-de-procesare ;

[12] C. Damodar, *Manual Testing Help*, 2012, Retrieved January 27, 2020, from https://www.softwaretestinghelp.com/manual-testing-help-ebook-free-download/comment-page-1/;

[13] D. Graham, E. van Veenendaal, I. Evans, R. Black, *Foundations of Software Testing: ISTQB Certification 1st Edition*, Cengage Learning Business Press, 2006.

Anca Ioana Andreescu graduated from the Faculty of Cybernetics, Statistics and Economic Informatics of the Academy of Economic Studies in 2001. She got the title of doctor in economy in the specialty economic informatics in 2009. At present she is an associate professor in the Department of Economic Informatics and Cybernetics of the Bucharest University of Economic Studies. Her domains of work are: informatics systems and business analytics programming languages.

Bologa Ana Ramona graduated from the Faculty of Cybernetics, Statistics and Economic Informatics of the Academy of Economic Studies in 1999. She got the title of doctor in economy in the specialty economic informatics in 2007. At present she is a professor in the Department of Economic Informatics and Cybernetics of the Bucharest University of Economic Studies. Her domains of work are: informatics