

Lightweight Model-Based Testing for Enterprise IT

Elodie BERNARD^{*†}, Fabrice AMBERT^{*}, Bruno LEGEARD^{*‡}, Arnaud BOUZY[‡]

^{*}*FEMTO-ST Institute, Univ. Bourgogne Franche-Comté, CNRS Besançon, France*

[†]*Sogeti, Lyon, France*

[‡]*Smartesting, Besançon, France*

[elodie.bernard|fabrice.ambert|bruno.legeard]@femto-st.fr — arnaud.bouzy@smartesting.com

Abstract—Model-Based Testing (MBT) popularity in IT is growing at a very slow pace. A recent survey stated that no more than 14% of respondents use MBT in their projects. Our experience, presented in this paper, demonstrates that the complexity in use of the current MBT approaches for the average tester is the main reason for this low dissemination. Then we introduce a lightweight MBT approach and a tool, called Yest, dedicated to business process-based testing of enterprise information systems. This tool uses a workflow-based graphical representation linked with decision tables to be used by functional testers without requiring any kind of modeling skill (such as UML for example). These approach and tool are dedicated to a particular class of applications (i.e. enterprise IT applications such as ERP and bespoke business applications). This focus strongly helps to simplify the approach and to adapt the tooling to the targeted users (namely IT functional testers). Finally, we discuss the way MBT may support emerging Acceptance Test Driven Development practices in agile.

Index Terms—Model-based-testing, Lightweight MBT, MBT in industry, MBT tool, ATDD, test generation, business process-based testing

I. INTRODUCTION

The Techwell Community report [1] shows that only 14% of the respondents (who are test professionals) use Model-Based-Testing (MBT). This low percentage can be explained by the current complexity and strong learning curve of the current MBT approach and tooling for the average functional tester [2]. By average functional tester, we consider a test professional, typically ISTQB - International Software Testing Qualification Board - certified at Foundation level. He/she understands well the business domain and test design techniques but without strong technical skill in software coding. He/she is in charge of test design, implementation, execution and test result analysis and reporting. Furthermore, such evidence of the complexity and strong learning curve of current MBT approaches and tooling are shown in the MBT User Survey 2016-2017 [3]: the average time required to be proficient in MBT is 185 hours for most of the respondents, and the median time is 100 hours. This is significant.

An important question for the future of MBT is about the simplification of the MBT approach and the adaptation of MBT tools to the targeted users (meaning the professional testers).

In this context, we propose a lightweight MBT approach with two clearly identified global objectives to attend.

- 1) Simplifying the modeling notation by focusing on targeted business domains - In our case, the Business Process-based testing for enterprise IT applications is targeted, thus we are using a simple workflow-based representation linked with decision tables.
- 2) Increasing the User eXperience of the MBT tool and adapting it to the user - For our modeling tool, a two-hour learning curve is the objective, by the continuous improvement of the tool GUI and features based on usability study with typical functional testers.

These two objectives are not the only one to increase the use of MBT. For example, a survey on testing methods used in the automotive industry [4] shows that MBT is used by more than 30% of the projects that have been analyzed. But, this is a very specific case where the model-driven development process is in use and is well established in the domain. Design models are then reused for testing purposes. In the enterprise IT applications testing context, models are usually not available ; sometimes even structured requirements are missing. Thus, testers need to cope with many missing information and still produce high quality test cases. To attend these global objectives, our work presented in this paper, is based on a collaboration between research and industry. In this paper, we focus on presenting the approach and the tool, and a first experience that contributed to reach the first level of these objectives. In section II, we present the workflow-based graphical representation linked with decision tables, that act as a DSML - Domain specific modeling language. In section III, we introduce the running example of the paper while in section IV, we present the test generation feature and the approach for test data definition. In section V, we present the support to test execution automation, before concluding in section VI.

II. WORKFLOWS AND DECISION TABLES

Currently the majority of MBT tools try to capture the expected behaviour of the SUT in various models [5]. Yest is a graphical tool for designing and implementing test cases developed by Smartesting. It has been designed for testing workflows and business rules. In Yest, the approach is to represent the workflow to be tested (and only what you want to test) and to keep it as simple as possible depending on your

test objectives. In figure 1, we illustrate the modeling elements used for representing the workflow.








Artefacts	Descriptions
	Start point: marks the start of the process
	End point: marks the end of the process
	Task: describes actions to execute on the SUT to test a behavior, and corresponding expected results
	Choice point: select the appropriate following flow depending on previous actions
	Sub process: creates a sub process
	Subset: defines a subset
	Connector: joins the different artefacts

Fig. 1. Artefacts of the DSML

There are three different kinds of inner nodes: task, choice point and subprocess. In the case of external nodes, one of them is the start point and the others are ending points. All nodes are linked by connectors to describe the flow. Task nodes describe actions on the SUT while choice points control the flow within the workflow. A subprocess is used to introduce a new process as a node of the current process. The subprocess has its own flow graph. It may introduce a hierarchical top down decomposition of the process that permits an easier manipulation of the workflow graph. Finally, the subsets are present to clarify the model by defining specific area of the workflow when necessary.

The following example (Figure 2) presents a model that uses the modeling elements of our DSML.

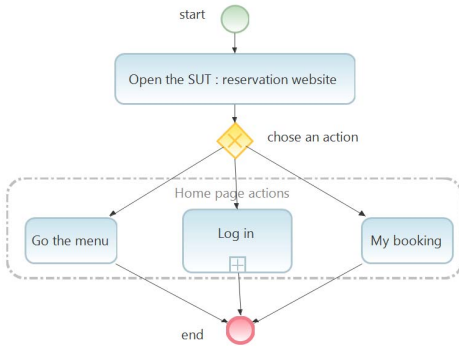


Fig. 2. Example of the use of Yest modeling elements

Within a workflow, the tasks and choice points are linked with decision tables to detail the cases to be tested and manage the flow for test generation, the abstract test data, the requirements traceability and the test step documentation (test actions and expected results).

ID	Identifier	ID	password	ID	Test steps	ID	Outcome	ID	Objective
1	correct		correct	1	Check the data The data are correct	check the data - correct data	Continue to the application		correct data
2	correct		incorrect	1	Check the data The data are incorrect	check the data - incorrect data	Connection error		correct identifier but incorrect password
3	incorrect		incorrect	1	Check the data The data are incorrect	check the data - incorrect data	Connection error		incorrect identifier and password

Fig. 3. Example of decision tables

A decision table is built with 5 types of column. First, "Test steps" is used to describe the test action (or sequence

of test actions) and related expected results (test step oracle) for one step. Then we may define 'Conditions' on test data set and 'Outcome' to control the flow. 'Requirements' and 'Objective' are complementary information to respectively trace the requirements and the related test objectives with a line in the decision table.

In the figure 3, the decision table's objective is to check if the data are correct or not in a context of a connection to an application. For checking the connection, there are three cases to verify:

- The connection data are correct
- The connection data are incorrect (correct identifier and an incorrect password)
- The connection data are incorrect (incorrect identifier and password)

There are therefore three lines in the decision table, each line representing a case to verify. It is possible to define several elements in the columns to guide the test cases generation, and to verify the requirements. Through the two firsts columns, we define the condition values for the identifier and the password. To have a correct connection and a redirection in the application, the identifier and the password must be correct. The identifier's value is therefore "correct" and the password's value is "correct". Thus the outcome of the test (visible in the Outcome column) will be "Continue to the application". In the same way, the error cases are defined through condition values with a correct identifier and a incorrect password, or an incorrect identifier and password. In both cases, the outcome will be "Connection error". Additional columns can be added, like the column Objective present in the figure 3, or other, in order to define some specific data like requirement, risk, priority etc.

III. RUNNING EXAMPLE

To illustrate this modeling approach, we introduce a small example related to a train ticketing system: oui.sncf. Oui.sncf is a French online travel agency (<https://www.oui.sncf>).

1) *Train reservation website under test*: To sum it up, this website is used to book train tickets with the possibility of creating an account on the website. To book a train ticket users can choose the departure and arrival cities and the date. To create an account, users have to enter a first name, a name, a date of birth, an email, and a confirmation email. With these information a set of requirements which need to be tested can be extracted.

2) *Requirements*: Here is a list of requirements that should be tested:

- It is possible to search a train between two cities at a specific date (Booking_requirement_010)
- In the train ticket reservation form it is impossible to choose a date departure in the past (Booking_requirement_011)
- In the train ticket reservation form, it is impossible to choose the same arrival and departure town (Booking_requirement_012)

- In the account creation form, if the information is valid, the account creation is possible (Account_requirement_010)
- In the account creation form, the format of the first name and the name have to be correct (no special characters)(Account_requirement_011)
- In the account creation form, the email and email confirmation have to be identical to create the account (Account_requirement_012)
- In the account creation form, the email should not already exist in the database to permit the creation of an account. (Account_requirement_013)

Figure 4 shows the workflow designed with the modeling tool in order to produce the test set covering these requirements.

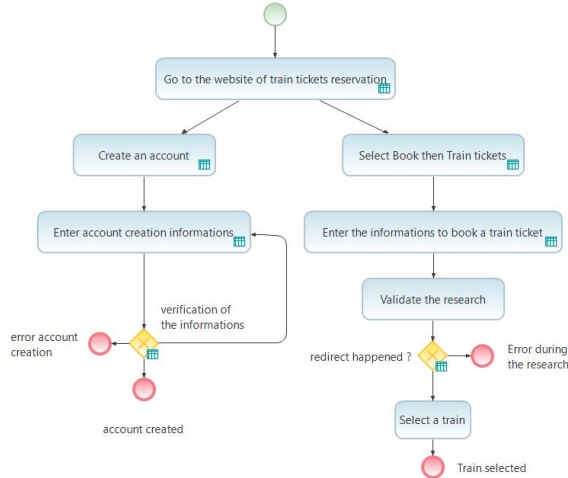


Fig. 4. Workflow of oui.sncf

The left part of the workflow represents the account creation and the right part the train booking. Let us consider the booking part. The process consists of a succession of four tasks. Starting by accessing to the train ticket reservation page, then entering the information, validating the research and finally, after checking the correctness of the information, selecting a train or dropping off. In the other part, for the account creation, an iterative process was used. The first step is the access to the account creation space, and then the completion of the information. The second step is to check if the information filled out allows the account creation. Once the behaviour is represented, the data can be chosen to verify our requirements and set up the requirements traceability.

The management of test data and of the requirements traceability of requirements is made through decision tables linked to tasks and choice points of the workflow. The figure 5 presents the table associated with the choice point to check the correctness of the account creation information.

	first_name	name	date_of_birth	email	confirmation_email
1	Smith	John	the 02/05/1994	john@mail.com	john@mail.com
2	???	John	the 02/05/1994	john@mail.com	john@mail.com
3	Smith	???	the 02/05/1994	john@mail.com	john@mail.com
4	Smith	John	the 02/05/1994	john@mail.com	smith@mail.com
5	Smith	John	the 02/05/1994	johnsmith@mail.com	johnsmith@mail.com

	Test steps	Outcome	Requirement
1	1 Check the informations	The informations are correct	-- account created Exigence_account_010
2	1 Check the informations	The first name is incorrect	-- Enter account creation informations Exigence_account_011
3	1 Check the informations	The name is incorrect	-- Enter account creation informations Exigence_account_011
4	1 Check the informations	The confirmation mail is different of the email	-- Enter account creation informations Exigence_account_012
5	1 Check the informations	The email already exists in the database	-- error account creation Exigence_account_013

Fig. 5. Decision table at node “verification of the information”

There are five lines, each of these lines represents a behaviour that needs to be tested to cover the different requirements. Thanks to the column *Requirement* a link is created between a test step and the requirement.

Green column headers indicate condition, that implies that the data sets are defined before (in the task *Enter subscription information*). Here the data are chosen according to the expected result. The account creation inputs are a first name, a name, a date of birth, an email and an email validation. With a combination of values associated with the condition columns, testers can activate one or the other of the SUT behaviour. So, in the second line of the table, when the first name has the value “???”, which is an invalid value, the process continues at *Enter account creation information* task and covers requirement *Account_requirement_011*.

You can observe the column *Outcome* that defines the outcome of the step, here the field of the account creation data is restarted. In this way the behaviour where users can redo the account creation while their data are not correct can be represented. Once the different tables have been filled, it is possible to generate automatically all the test scenarios covering all lines of all the tables (or only the selected lines). After the generation of test scenarios, several metrics are accessible, notably the requirements coverage.

3) *Control of the requirements traceability*: A dashboard provides a coverage overview (how the control flow and lines in the decision tables are covered by the generated test scenarios, and the coverage of added information in the tables: requirements, risks, test objectives), and traceability between those added information and the test scenarios. It is possible to find all the requirements that were defined in the tables, and the scenario(s) that cover(s) each of them. In this way the validity of our model can be checked as well as the requirements coverage.

IV. TEST GENERATION

Test generation facilities are based on two main principles:

- *Principle 1 - Let the user keep the control over MBT test generation*. This means that the tool should propose test scenarios, according to the workflows and decision tables, but the user may be able to change everything in these proposed scenarios (meaning the test steps, test actions and expected results descriptions, and test data). He/she should also easily create new test scenarios in

relation with the workflows and decision tables. The tool permanently provides information about the compliance between the test scenarios and the models, but the user may chose to keep any test scenarios, even if it is not compliant with the current state of the model. The MBT tool should never be a blocker or barrier to create required test scenarios, only a support.

- *Principle 2 - Do not provide test selection criteria difficult to understand by the user.* We have seen on previous experience with MBT tools [6] that classical MBT test selection criteria such as condition/decision coverage [7] may be difficult to understand by functional testers. Moreover, the main test methods in use in enterprise IT for functional testing is requirements-based testing (RBT), meaning the coverage of the requirements by the test cases. The MBT tool should first support this approach, proposing, thru automated test generation, the test scenarios that are covering the requirements linked in the decision tables. This approach should also cover any other type of information such as risks, priorities and test objectives, that can be added in the decision tables.

In the next two subsections, we describe how these principles are implemented in the tool, and supported in two ways: by automated test generation and by supporting the interactive creation of a new scenario matching a workflow and linked decision tables.

A. Automated test generation

Automated test generation is based on covering the lines in the decision tables. The user may run automated test generation at the level of the workflow (meaning covering all tasks and all lines of the decisions tables in the workflow), or at the level of each decision table (to cover all the lines of a decision table or the selected lines). During automated test generation, the various constraints on test data that appears in the lines of the decisions tables (in the columns 'Conditions') are solved (using constraint solving techniques). Each test scenario resulting for test generation can be visualized graphically on the workflow (Figure 6).

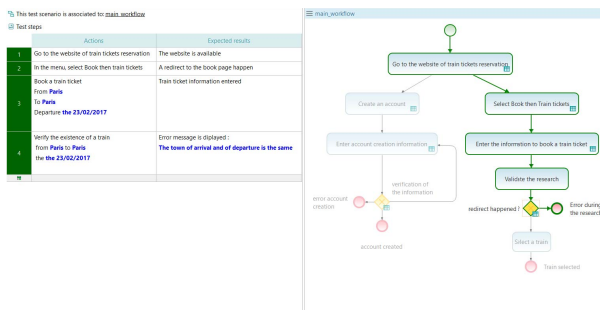


Fig. 6. Example of generated test

The user can edit each generated test scenario, updating the test data values, test step description or also the test step. Each time some change is done on a scenario, the matching

between the updated scenario and the workflow is performed and displayed to the user. If there is no matching, the user may keep the scenario as it is, or update it, or update the workflow and decisions tables.

The name of each test scenario is computed automatically by concatenation of the test objective reported in covered lines of the decision tables. But the name may also be changed by the user.

When some part of the workflow or some lines in decision table are not covered, the tool provides information about the possible cause on the missing coverage. This can be for example because no matching solutions can be found regarding test data values to satisfy the constraints defined in several tables. Then the user may adapt / change it in order to cover the missing part.

The generator can bring to light the uncovered part of the model. A common mistake, when using the outcome of the tables, is to forget to specify an issue. So, if a part of the model is uncovered, some corrections specifying the missing outcome need to be added, and the generator will cover the model. Thanks to the different systems of detection of incoherence in the model, the generator can produce a set of relevant scenarios to verify the requirements by ensuring the lowest possible number of scenarios. However, it is possible to easily produce custom scenarios thanks to the *assistance of workflow compliance*.

B. Interactive creation of a scenario

The interactive creation of a scenario is guided by the *assistant of workflow compliance*. This allows to produce a step by step test scenario, compliant with the workflow and linked decision tables. Auto-completion helps to create each step of the targeted test scenario. At each step, the tool checks the conformity of the scenario. Producing the scenario by this interactive way is also a good technique to validate the model and help users to focus on test objectives.

In the same way, the assistance can help users to create a scenario. At each step it is possible to select the line in the tables of the model that needs to be covered, and with each new action, multiple scenario completion are proposed.

Thus, progressively, in the creation of the scenario, it is possible to follow the requirements coverage through the model. Then, the scenario is progressively created and the model coverage checked. The assistant is visible in figure 7.

C. Test data

A good practice in MBT (see [7]) and in software testing in general, consists in separating test design from test implementation:

- Test design aims to produce abstract test cases - i.e. test cases without concrete (implementation level) values for input data and expected results.
- Test implementation aims to produce concrete test cases (or test scripts in case of test execution automation) from abstract test cases. Concrete test cases are test cases with concrete (implementation level) values for input data and

	Actions	Expected results
1	Go to the website of train tickets reservation	The website is available
2	Create an account	The account creation form is available

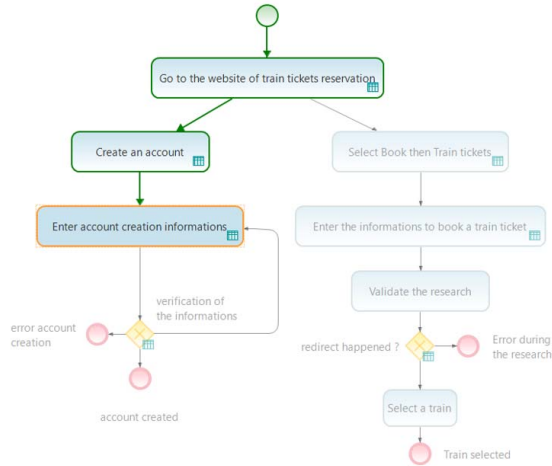


Fig. 7. Assistant of workflow compliance

expected results, and with very detailed descriptions of test steps.

The generation of abstract test cases is supported by our MBT approach and MBT tool. The implementation of the test cases for manual test execution is supported by the refinement of test actions in detailed steps (this is supported by the tool). In case of test execution automation, test automation artifacts (test scripts, adaptation layer skeleton) are automatically produced using publishers adapted to a targeted test automation framework (see next section).

In the decision tables, test data are generally defined as abstract test data, representing equivalence partitioning [8] with some values for each partition on a dedicated line of the table. If he/she prefers, the user may use some concrete data in order to simplify the implementation phase. A dictionary of all data defined in the model is provided to sum-up data types and values in use in the decision tables - cf. figure 8

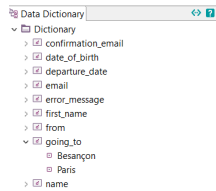


Fig. 8. Data dictionary

Once generated, the test cases must be implemented to be ready for test execution. The MBT tool may convert them into the test automation scripting language of the automation test execution tool. The MBT tool may also publish them as documented test cases, including traceability with requirements into a test management tool for manual test execution.

V. TEST AUTOMATION

For automated test execution, the test cases must be generated in a form that is executable. Our approach is based on the well known Keyword-driven automation best practices [9]:

- The MBT generated test cases are converted into automated test scripts, where the test actions described in the model are converted in test automation keywords with parameters (the test data). The format of the scripts depends of the test automation framework in use. For example, for a web application, this may target Java Selenium and then the scripts are published in Java.
- A test adaptation layer code is written by test automation engineers to bridge the abstraction gap between the test actions from the model and the execution interface (i.e. the keywords) on the SUT. The automated execution may be GUI-based or API-based (or based on web services). This code is essentially a wrapper around the system under test in order to manage test execution.
- In order to variabilize the test scenario at execution time, a data-driven approach is supported - i.e. “A scripting technique that stores test input and expected results in a table or spreadsheet, so that a single control script can execute all of the tests in the table” cf. [10]

For the running example, the Selenium-Java publisher of the modeling tool was used. The principle of this add-on is to take the scenarios produced by the model and convert them into test scripts. In the same way, the abstract data sets in the scenarios are transformed into concrete data in the scripts. Each scenario uses actions, and an action can be common to many scenarios. So to generate the test scripts, the tool supports the mapping of each test action with a concrete automation keyword.

The transformation of the actions consists in defining the automation functions (Keywords) needed for automated test execution on the SUT. These Keywords are defined in an Excel file through a table, which includes the Keywords names and their parameters. An example of Keywords definition can be seen in the figure 9. The third line of the table contains the definition of the Keyword “enter_subscription_information”, linked to five parameters: the first name, the name, the date of birth, the email and the confirmation email. The first column define the package name where the Keywords file will be placed. Once the Keywords have been created in the table, it can be imported in the modeling tool to complete actions in the interface of actions completion.

Class	Keyword	param1	param2	param3	param4	param5
Keywords.Keywords	book_a_train_ticket	from	to	departure		
Keywords.Keywords	check_the_information	errorMessage				
Keywords.Keywords	enter_subscription_information	firstName	name	dateOfBirth	email	confirmationEmail

Fig. 9. Example of Keywords definition

In figure 10, the completion of the action “Enter subscription information” is illustrated. The associate Keyword is “enter_subscription_information” as seen above. The table at the bottom of the interface shows the chosen Keywords, with the list of parameters names in the columns headers. The

completion of the parameters value is done in the row, in front of the keyword name.

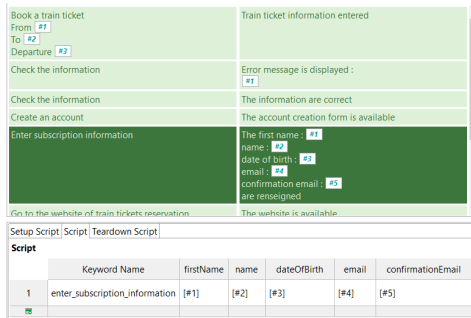


Fig. 10. Add-on of automation in Yest (for Selenium Java)

The values of these parameters can derive directly from the data of the scenarios (when we use #1, #2, etc.). It is the case when the abstract data in the scenarios correspond to the concrete data in the SUT. Thus the tester has the possibility of choosing data directly from the scenario (#1, #2, etc) or define new data if the automation adaptation layer needs it. To define new data, the required data just have to be written in the linked cell. If a different name than the one of the existing value is needed then [#2] can be replaced by the expected value. This system can be useful when it is necessary to manipulate only the data required for test execution automation.

Another possibility is to call several Keywords for the same action. In this way the action in the model can be very abstract to keep the model simple. Once these tacks are completed, the script can be produced: the scripts reflect the scenarios and are composed of a sequence of Keywords.

The approach facilitates the maintenance of the automation code. Let's consider two situations:

- Some changes have been made in the interface of the application, the test script fails. These changes impact the Keywords but not the model. Here, we have to modify the implementation of the relevant Keywords to maintain the test scripts.
- Some changes have been made in the requirements, and the changes impact the model. These changes can be the addition of data, modifications of the test workflow. This may bring the update of existing Keywords or the creation of new Keywords.

The clear separation between the test scenarios (generated from the workflows and the tables) and the technical implementation of the keywords help to manage the maintenance of the automation code.

VI. CONCLUSION: MBT FOR ATDD IN AGILE

As we discussed earlier, the adoption of MBT by industry is progressing slowly, and at the same time, the digital and agile transformation trends increase the need for better and more automated software testing methods. As stated in the last edition of the World Quality Report (2017-2018) [11]: "Combined with the increased speed of deployments and growing

complexity of the application landscape, these trends increase the risk of introducing serious errors and software failures. Furthermore, the end-customer today has zero tolerance of errors or slow performance, and poor quality of software can result in serious damage to the brand value of an organization and often incurs huge repair costs".

We believe that re-designed MBT approach and tool may strongly help test teams to face the transformation to agile and the increased complexity of systems by supporting ATDD - Acceptance Test Driven Development [12]. ATDD involves team members with different perspectives (customer, development, testing) collaborating to write acceptance tests in advance of implementing the corresponding functionality. The visual test design supported by MBT (i.e. by using graphical workflows and decision tables in our approach) strongly helps the collaborative discussions between Business Analyst, Product Owner, Testers and even Developers to better understand the business requirements to be implemented. Then, these MBT-generated acceptance tests represent the user's point of view and act as a form of requirements to describe how the system will function, as well as serve as a way of verifying that the system works as intended.

But, to efficiently support ATDD, the MBT solution should satisfy several requirements:

- *Short learning curve and good usability by functional testers.* The MBT approach and tool should easily be adopted by existing functional testers. We have seen from the MBT User Survey 2016-2017 [3] that the number of hours to be proficient in MBT is on average of the respondents 185 hours, and the median value is 100 hours. This should be reduced to a couple of hours. This means a drastic simplification of the modeling concepts and an adapted tool support.
- *Adaptation of MBT to iterative and incremental development approaches.* The adaptation to agile means that MBT should be efficient for various scope and test objectives, such as to test a small set of new user stories, to keep alive a end-to-end test set capturing the main use cases of the applications, and to capture a specific usage scenario that create issues in operation. Therefore the MBT approach and related tool should be very flexible and never required full modeling of the application to support test production.
- *Supporting both manual and automated test execution and a seamless adaptation from the first one to the second one.* Manual test execution of acceptance tests by testers during the agile iteration is generally the first outcome of MBT-generated tests, then the test team needs to extend the regression test repository. The MBT tool should support both usage and facilitate / automated the adaptation of manual test to automated tests.

We described in this paper a lightweight MBT approach and tool called Yest. Yest is intended to support the functional test designer by taking advantage of graphical workflows representation. It has been designed for testing business processes

and business rules of enterprise IT applications, with a very short learning curve. In this way, it differs classical MBT tools, often complex to learn for testers in industry but more general in the targeted domains.

The first experiences on large-scale enterprise IT projects by several test teams (functional testers) in agile delivery show that this requirements tend to be satisfy by the proposed approach and tool. The process starts by two-hours hand-on session. Then the test team start test generation of a first limited scope with 3 or 4 on-hour revue sessions to discuss good practices for modeling and generating (particularly to advocate and show how to keep the workflows as simple as possible). The level of details to test documentation depends of the project context (i.e condition of execution), but the general idea is to draw the workflow with the minimum of details required by the test objectives, test execution methods and sharing whit stakeholders. The most important is the improvement in efficiency obtain by the test team by clarifying the business requirements, visualizing the coverage of the workflows and requirements (through traceability) and increasing the global productivity of the test team.

We are currently working on an experimental protocol to measure on several projects the gains based on metrics (to be compared with manual test design approach) such: the number of generated test cases / scripts, and the number of generated test cases / scripts per person-day, The number of defects found in the requirements during MBT modeling activities, or the reusability level of MBT model elements from one project to another.

REFERENCES

- [1] *The state of the Software Testing Profession 2016-2017*, Techwell Community. [Online]. Available: https://stickyminds.com/sites/default/files/webform/file/2017/16-17_SotTP_report.pdf
- [2] S. Ali and H. Hemmati, "Model-based testing of video conferencing systems: Challenges, lessons learnt, and results," in *Verification and Validation 2014 IEEE Seventh International Conference on Software Testing*, pp. 353–362.
- [3] A. Kramer, B. Legeard, and R. V. Binder, "2016 MBT user survey." [Online]. Available: <https://www.researchgate.net/project/2016-MBT-User-Survey>
- [4] H. Altinger, F. Wotawa, and M. Schurius, "Testing methods used in the automotive industry: Results from a survey," in *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing*, ser. JAMAICA 2014, New York, NY, USA: ACM, 2014, pp. 1–6. [Online]. Available: <http://doi.acm.org/10.1145/2631890.2631891>
- [5] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," vol. 22, no. 5, pp. 297–312. [Online]. Available: <http://onlinelibrary.wiley.com/doi/10.1002/stvr.456/full>
- [6] B. Legeard and A. Bouzy, "Smartesting certifyit: Model-based testing for enterprise it," *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pp. 391–397, 2013.
- [7] *ISTQB Foundation Level Certified Model-Based Tester Syllabus*, International Software Testing Qualifications Board, 2015. [Online]. Available: <https://www.istqb.org/downloads/syllabi/model-based-tester-extension-syllabus.html>
- [8] L. Copeland, *A Practitioner's Guide to Software Test Design*. Norwood, MA, USA: Artech House, Inc., 2003. [Online]. Available: http://www.dahlan.web.id/files/ebooks/2004%20A%20Practitioner%27s%20Guide%20to%20Software%20Test%20Design_Good.pdf
- [9] R. Hametner, D. Winkler, and A. Zoitl, "Agile testing concepts based on keyword-driven testing for industrial automation systems," in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, Oct 2012, pp. 3727–3732.
- [10] "Istqb glossary." [Online]. Available: <http://glossary.istqb.org/search>
- [11] "World quality report 2017-18," Tech. Rep., 2017-2018. [Online]. Available: https://www.sogeti.com/globalassets/global/downloads/testing/wqr-2017-2018/wqr_2017_v9_secure.pdf
- [12] M. Gärtner, *ATDD by Example: A Practical Guide to Acceptance Test-Driven Development*. Addison-Wesley, 2012.