A Methodology for Enterprise Resource Planning Automation Testing Application to the Open Source ERP-ODOO

Thierno Birahime Sambe[2][a], Stephane Maag[1][b] and Ana Cavalli[1][c]
[1]*Institut Mines-Télécom/Télécom SudParis, Université Paris-Saclay, CNRS UMR 5157 Samovar, France*
[2]*Institut Supérieur d'Informatique, ISI-DAKAR, Dakar, Senegal*

Keywords: ERP, Business Processes, Automated Testing, Test Generation, Test Execution, Business Driven Testing.

Abstract: This paper presents an approach for the automated testing of Enterprise Resource Planning (ERP) systems. ERPs are complex software systems that provide a chain management covering all corporate activities. Testing of these systems is a complex task given the close interrelation between the functionalities of system modules. Test automation is also an important issue, in order to reduce testing time and cost. To this end, our approach for the automation of ERP systems testing, is based on the system modeling as a set of business processes to meet the needs of the business and reduce, for this purpose, the risks of errors. The methodology used in this context is to combine the system modeling and the tools to manage automation of ERP-tests. Following this approach, from the basic requirements, a number of test purposes are defined and test cases are generated using a test generation tool, which are then run on the ERP system. In order to illustrate the application of the proposed method, test experiments have been performed on a real case study, the ERP ODOO.

1 INTRODUCTION

Enterprise Resource Planning (ERP) Systems are complex bundle of software that are well-integrated in such a way that they present a one stop information bay for all operations in the organization. They encompass many modules and with every module integration comes a challenge and an opportunity for testing (Nagpal et al., 2015). Enterprise Resource Planning (ERP) system is an information system that helps the organization to coordinate and integrate information within departments (Chen et al., 2016).

ERP software automates core corporate activities, such as manufacturing, human resource, finance, and supply chain management, by incorporating best practices to facilitate rapid decision-making, cost reductions, and greater managerial control (Holland and Light, 1999).

As stated in (Nagpal et al., 2015), there is an exponential rise in complexity of software testing with the increase in the number of modules implemented. With the rise in the implemented modules, functional complexity increases, there is an increase in the interfaces between the modules that lead to an increase in the user authorizations and access rights that further aggravate the testing complexity of ERP implementations. Many of the ERP Implementation projects fail because of the improper test planning and testing process. To understand the importance of ERP testing, testing professionals need to understand the variations of the ERP systems (Software-Testing-Help, 2018).

The software vendors have their own methodologies for testing, but these focus on process flows through configured components. Such tests prove the software "does what it does" (Gerrard, 2007).

ERP Testing is really critical if the testing plans do not cover the complete road-map of the implementation phase. All the modules within the system are completely integrated and interrelated, it means that all the functionalities are dependent on the functionalities of other modules or other systems. The main challenge is in collecting input data for testing the feature and analyzing the correctness of the output data, which requires domain knowledge (Software-Testing-Help, 2018).

The customization that goes along with an ERP system, plus the amount of complex business functionality involved, can make testing ERP systems a challenge of strategic importance. ERP is not just a software, it is also a business process. It needs a dedicated workforce to maintain and test it. Given the business orientation of most of ERP tests along with the repetitive reruns of tests necessary when there are new releases, testing can benefit greatly from mainte-

[a] https://orcid.org/0000-0002-6594-1077
[b] https://orcid.org/0000-0002-0305-4712
[c] https://orcid.org/0000-0003-2586-9071

Sambe, T., Maag, S. and Cavalli, A.
A Methodology for Enterprise Resource Planning Automation Testing Application to the Open Source ERP-ODOO.
DOI: 10.5220/0007923004070415
In *Proceedings of the 14th International Conference on Software Technologies (ICSOFT 2019)*, pages 407-415
ISBN: 978-989-758-379-7

nance friendly automation (Christine Paras, 2018).

To tackle the shortcomings of these traditional approaches, there is a growing need for solutions to automate testing processes. Test automation is an ideal approach to test software quality across regression cycles with reduced efforts, time and cost. With these evolutions from time-to-time, test automation has not just remained a traditional testing practice, it has evolved to become a mature, business-driven testing (BDT) program that ensures better test-coverage, scalability and productivity in the entire testing efforts to deliver quality software applications and systems (TestingWhiz, 2017). To this end, our approach, for the automation of ERP systems testing, is based on the modeling of business processes to meet the needs of the business and reduce, for this purpose, the risks of error compared to business processes initially expressed by the top level "top management". The methodology used in this context is to combine a set of modeling methods (high-level business processes) and tools to manage automation of ERP-tests. In this test automation approach, the top management team and business analysts intervene right from the start of the project to define business rules and test cases and generate automated scripts, which are then run on the application.

Our paper is organized as follows. Section 2 presents the related work and Section 3 introduces the basic concepts and the tools that will be used by our testing approach. Section 4 presents our test automation approach based on the modelling of the ERP business processes. Section 5 presents the results of the test experiments we have performed on a real case study: the ERP ODOO. Finally, Section 6 concludes this work.

2 RELATED WORKS

Testing processes have been worked for a while and there exist several proposals in the literature for ERP Testing. We present in the following the ones we get inspired.

Schliesser (Schliesser, 2007) points out that traditional software testing methods are not sufficient and not designed for meeting the challenges posed by ERP implementations. He condescends that built in mechanisms and documentations available from the ERP vendors can be utilized to effectively plan meaningful business test cases.

Gerrard (Gerrard, 2007) has noted the lack of research in ERP testing. Few books that exist are more focused on product centric testing rather than on wholesome ERP testing, thus software 'does what it does'. The authors bring forth that testing in ERP implementations is critical, but is often neglected. They have presented ground work to define test methods and tools specific to ERP implementations.

Wieczorek et al. (Wieczorek et al., 2008) discussed that software development and testing of ERP systems require methods that are dedicated to tackle the special features of ERP systems. The complex ERP systems make manual testing cumbersome, and even effective automated testing requires appropriate test data. The authors came up with challenges and possible solutions for the test data required for automated testing in case of ERP systems.

Wieczorek and Stefanescu (Wieczorek and Stefanescu, 2010) point out that system level testing is the dominant testing approach in case of ERP systems. Recent shift of ERP systems towards SOA has led to the demand of black box testing techniques. They advocate more research in system testing with specific focus on user interfaces in ERP systems.

In (Downing, 2006), a comprehensive report on twelve mistakes that should be avoided is written. The authors also depict Best Practices that need to be adopted in order to manage quality in an ERP implementation. Ensuring quality itself is a team effort from the ERP vendor, implementation partner, hosting provider, technical and business teams.

Deanna Legge (Legge, 2011) underlined the need for structuring a robust testing strategy in her paper on ERP testing methodology approach. She emphasized that since ERP systems are complex that encompass several integration points, geographies, customizations, etc.; organizations continuously face challenges to deliver quality ERP solutions.

3 BASICS PRELIMINARIES

3.1 Extended Finite State Machine – EFSM Model

This section introduces the notations related to the definition of an Extended Finite State Machine. As defined in (Maag et al., 2008), an Extended Finite State Machine (EFSM) is an ordinary Finite State Machine (FSM) augmented with context variables, input/output parameters, predicated and update functions. Given input and output alphabets I and O, we denote by Ri the set of input parameters i I, and by Qo the set of output parameters for o O. We also denote by R the union of Ri over all i in I and by Q the union of Qo over all o in O. The finite set of context variables is denoted by V. As usually, we use N to denote

the set of natural numbers. An Extended Timed Finite State Machine, in short ETFSM, is a tuple M = (S, s0, I, O, T, δ, v0) where:

- S is a non-empty finite set of states with the initial state s0;
- I and O are input and output alphabets;
- T is the set of transitions;
- $\delta : S \longrightarrow S \times \{N \times \{\infty\}\}$ is a timeout function; and v0 is the vector of initial values of the context variables

3.2 IF-Language

3.2.1 IF-Model

IF is a formal method based on communicating timed automata in order to model asynchronous communicating real time systems (Bozga et al., 2002). In IF, a system is expressed by a set of parallel processes communicating asynchronously through a set of buffers.

A process instance can be created and destroyed dynamically during system execution. An IF process is described as a timed automaton extended with discrete data variables. A process has a set of control states and a private buffer for input messages, and can have local data, such as discrete variables and clocks. There are two types of control states: stable states and unstable states. An unstable state is a temporary state where no interleaving between processes is possible. In other words, if a process moves to an unstable state by an action, the atomicity of the execution is guaranteed until it reaches a stable state.

Transitions describe the behavior of a process on stimuli. A transition can be triggered either by (timed) guards or by an input message where an urgency attribute (eager, delayable or lazy) defines the priority of the transition execution over time p.

When an eager transition is executable, time progress is blocked until the transition is executed. If there is an executable delayable transition, time can progress as long as the transition is executable. If time progress makes the delayable transition non-executable, time progress is blocked until the transition is executed. For lazy transitions, time can progress although the transitions become non-executable. The action of a transition may include sending output messages, setting/resetting clocks, assignment of variable values, and creation/destruction of processes.

3.2.2 IF Toolset

The IF toolset (Bozga M, 2004) provides an environment for modeling and validation of an IF specification. The core components of the toolset are the IF static analyzer and the IF exploration platform. The IF static analyzer transforms an IF specification into an abstract syntax tree which is a collection of C++ objects. The IF exploration platform performs the simulation of process executions using the abstract syntax trees. A set of APIs is provided by the IF exploration platform, which allows implementation of user-specific exploration.

In the IF toolset, it is possible to check if given properties hold for an IF specification using observers. Once a property is described in the IF language using a specific syntax for observers, e.g. monitoring of events and cutting off generation of irrelevant states, it is executed in parallel with the target system.

3.3 Selenium Webdriver

- Selenium

Selenium IDE is a one of the most popular free open-source automated testing tool which provides a testing framework for testing web applications and supporting multiple kinds of frameworks. It can be easily downloaded from internet as a plug-in for some browsers. It is basically used by the web development community to perform automated testing of web applications. We choose in our study Selenium Webdriver because Selenium IDE does not support record-playback feature and also it is most supported for web-application testing (Monier and El-mahdy, 2015).

Selenium is a web testing tool which uses simple scripts to run tests directly within a browser (Holmes and Kellogg, 2006). It uses JavaScript and iframes to embed the test automation engine into the browser. This allows the same test scripts to be used to test multiple browsers on multiple platforms. Selenium gives the user a standard set of commands such as open (a URL), click (on an element), or type (into an input box); it also provides a set of verification commands to allow the user to specify expected values or behavior. The tests are written as HTML tables and run directly in the browser, with passing tests turning green and failing tests turning red as the user watches the tests run. Because Selenium is JavaScript-based and runs directly in the browser (the user can see the test running), it overcomes some of the problems encountered by users of HttpUnit or Canoo WebTest, particularly problems related to test-

ing JavaScript functionalities. Two additional useful tools are also available for use with Selenium: 1. the Selenium IDE (originally called the Recorder), which allows users to navigate their applications in Firefox and record their actions, forming tests; and,

2. a “Remote Control” server which allows users to write test directly within the programming language of their choice – thus enabling conditional logic within tests, try/catch blocks, and other powerful functionalities available only within programming languages.

Selenium allows the identification of elements using the browser’s DOM object, the test can be written using specific identifiers of the necessary element, such as name, id, or xpath (Holmes and Kellogg, 2006) (Table 1).

Table 1: Input data for a Selenium Test.

type	name=theField	Text to submit
clickAndWait	id=SubmitButton	
assertText	xpath=//h1/span	Success!

This test might be written for an HTML page as simple as (Fig. 1):

```
<html>
<input name="theField">
<input id="SubmitButton" type="submit">
<h1><span>Success!</span></h1>
</html>
```

Figure 1: HTML test cases in selenium.

When the test is run, each command is highlighted as it executed, and the assert steps turn red or green to indicate success or failure. After the whole test is complete, the test is marked as red or green in the Suite.

- Robot Framework

Robot Framework (Pajunen et al., 2011) is a generic keyword-driven test automation framework meant for acceptance level testing. The tool uses keyword abstraction for test design. Keywords are divided into higher-level user keywords and lower-level library keywords. The user keywords are built by creating combinations from the keywords presented in the keyword libraries. Test cases are described with scripts that use the keywords and control structures, such as loops. The available keywords of Robot Framework are defined in libraries. New libraries can be implemented with Python or Java. There are two types of libraries: standard and external. The standard libraries are distributed with Robot Framework and the external libraries are released in separate packages:
Standard libraries

- BuiltIn: Keywords for generic testing needs, such as variable verification, conversions and delays.
- Operating System: Keywords for operating system tasks,such as file system operations and executing commands.
- Telnet: Keywords for Telnet connection handling.
- Collections: Keywords for handling lists and dictionaries.
- String: Advanced string handling keywords.
- Dialogs: Keywords for getting user input during test execution.
- Screenshot: Keywords for capturing and storing screenshots.

4 ERP TEST AUTOMATION: APPROACH BASED ON THE MODELING OF HIGH LEVEL BUSINESS PROCESSES

4.1 Approach based on Business Process Modeling (Business Driven Testing)

The problem to be solved is how to manage the automation of ERP systems tests, based on the modeling of business processes to meet the needs of the business and to reduce, for this purpose, the risks of error compared to business processes initially expressed by the top level ”top management”.

The methodology used, in this context, consists of combining a set of modeling methods (high-level business processes) and tools to manage test automation of ERP, which involves the following different steps:

Step 1: We use Business Driven testing (BDT) to Model the High Level Business Processes defined by the Top Management; which we have called ”HLBP” in our architecture, to contribute to approach based on business needs. **Behavior Driven Tests** (BDT) focusing on high level design, not on the technical implementation nor the technical terminology. Stakeholders or Users, Business Analysts, Testers, Developers will involve and perform their Respective roles. Users and business Analysts participate in test cases review process and give their feedback to enhancement. Behaviour driven tests (BDT) are easier to modify and hence. It is then very easy to maintain.

In this step, it is a question of referencing (gathering) all the Business Processes defined by mutual

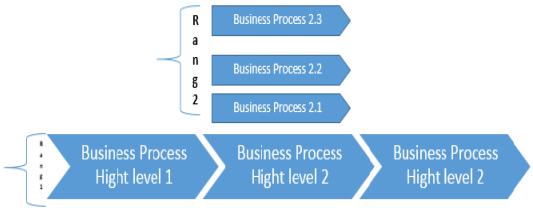

Figure 2: Level Business Process Model.

agreement between the various stakeholders. Then we proceed to the tree of these processes starting with those of rank 1 (high-level process) then by the processes of the following ranks. At this stage, the following elements will be retained (illustrated by Figure 2:

- Objectives of the process
- Purpose of the process
- Main actor(s) of the process

Step 2: Explode all Rank 1 processes (high level) in Tier 2 processes that will cover all processes and behaviors of the business processes of Rank 1. We will apply the Behavior Driven Testing (BDT) from model based testing by using the IF-language. The automation of tests generation from a formal model, also known as a formal model-based testing, was integrated into the software development process and was adopted in this step in our test automation architecture. This technique is known as model-based testing and it refers to the ability to detect failures which are not conform to a model. There are several approaches used for model-based testing that include finite state machines -FSM- and labeled transition systems -LTS- with several extensions and variations. The use of finite state machines is the traditional technique used for testing, originating in the domains of communication protocols and sequential circuits. The formal model proposed for ERP-Testing Automation is based on finite state machines augmented with time and parameters, allowing the representation of basic units of the real-time system. In order to represent temporal requirements, timeouts are used since they allow the easy modeling of some internal critical issues of the system. The Figure 3 illustrates the concept of a finite state machine with timeouts.

The fundamental components of finite State Models include:

- **State** represents the “mode of being” for the system at any given time.
- Transition describes a single pathway from a state to another state. The set of all Transitions describe all possible paths among the defined states. A Transition contains an event that it is subscribing

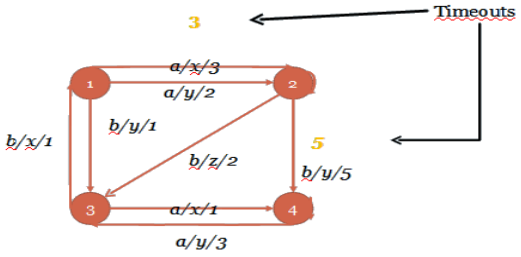

Figure 3: Finite state machine with timeouts.

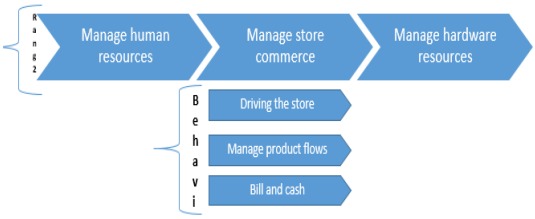

Figure 4: Tier 1 ”Manage Store Trade” process.

to (the event that triggers execution of the Transition), a Condition and an Action. A Transition also contains references to the state from which the Transition is exiting (i.e. the “from” state) and the state to which the Transition is entering (i.e. the “to” state).

- **Event** is the mechanism that the system uses to interact with external systems and with itself.
- **Condition** represents a set of logic that evaluates to a Boolean result. A Condition is used to determine if a Transition is to be executed.
- Action is the logic to be executed when a Transition is executed.

Example: for an ERP of commercial management of a store, where the top management decides to ”Manage the store’s trade”. We define all processes and activities in the Tier 1 ”Manage Store Trade” process (Fig. 4).

- Step 3: it consists in defining the test objectives emanating from the activities and behavior of the business processes.

  A test objective describes a particular functionality of the implementation under test, by specifying the property to be checked in the system implementation. It is an observable action of the system that once described in IF-language is used for guiding the space exploration of the system’s states. A test objective is described as a conjunction of conditions, including optionally the following: an instance of a process with an identifier, a state of the system (a source state or a destination state), an action of the system (a message sent, a message received, an internal action),

a variable of the process or a clock of the process, specifying a value and its state (active or inactive). A test objective obj is formally described as in the figure below (Figure 5):

```
obj = cond1 ^cond2 ^ : : : ^condj
condi = process : instance = {proc}id |
condi = state: source = s |
condi = state: destination = s |
condi = action : input signal(parameters) |
condi = action : output signal(parameters)|
condi = variable : v = value|
        condi = clock : c = value |
condi = clock : c is active/inactive
```

Figure 5: Test-objective structure.

- Step 4: derivation of objective, measurable and manageable test objectives with the TestGen-IF tool (hwa, 2009). The tool was defined and implemented for modeling and simulating asynchronous timed-systems such as telecommunication protocols or distributed applications. It is based on active testing techniques, allowing automatic generation of test cases from a formal description of the studied system. The generation is made according to specific objectives called test purposes, which are described more in detail in the next section.

  TestGen-IF tool allows to construct the accessibility graph from an IF specification. Therefore, the inputs necessary for TestGen-IF tool are the formal functional specification of the system and the specification of test objectives that we wish to check on the system implementation. Then, the tool makes a partial exploration of the states space of a model, guided by the tests objectives (Fig. 8).

  Illustration 1: Example Process Model: in Sale Module on ERP

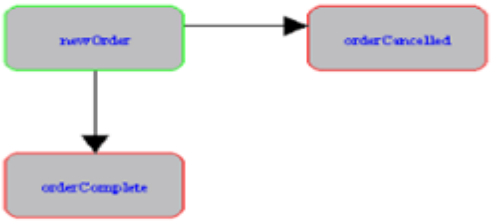

Figure 6: Process Model : in Sale Module on ERP.

In this example (Fig. 6), we will build an overly simplified control process. The order process is: order received, inventory control, cancel the order if the product is not in stock. Otherwise, complete the command.

Illustration 2: Example Process Model Processes and Asynchronous Events : Unlike our first example which showed a very simple order management process. In this example (Fig. 7), the process is short-lived and synchronous - as soon as it is started, it runs without any dependency on incoming events. The second example is similar to the first, but with added asynchronous behavior. In this example, the process waits and responds to asynchronous events that indicate changes to the inventory. When the command is created, if the stock is available to execute the command, the command is terminated normally. Otherwise, the process waits until it receives a notification that the appropriate inventory is available when the process is complete. The new state template is shown below:

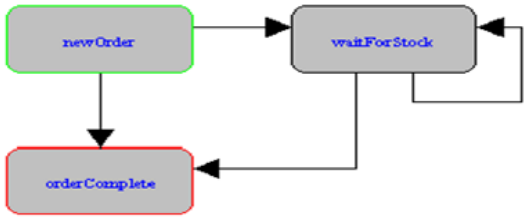

Figure 7: Sale Order: in Sale Module on ERP.

- Step 5: It translates the Objectives into the Automation Tool "Selenium" with WBA (Web Brother Automation) for the automatic execution of test cases in ERP;
- Step 6: A verdict is provided by the system to confirm if the tests are "OK" or "NOK"

5 EXPERIMENTS: APPLICATION TO THE ERP-ODOO

Odoo is not simply an ERP software shipped by Odoo SA. It is an open source project with an important community of partners, clients, individuals or companies, who believe in the value of sharing: code of course, but also tips and ideas.

How to make and execute the tests: Recording actions using Selenium :

- We install Selenium on Odoo (Selenium, 2018). This addons adds tags in the XML of Odoo that help Selenium recorder to understand the HTML structure and record it. .
- We install selenium recorder on Firefox (Builder, 2018). This addons adds in your browser an option to "start recording"|"stop recording".
- **1**. Add context in HTML code : first, the web-selenium Odoo module generates semantically richer HTML elements. By using Odoo's template extending mechanism, this module adds the

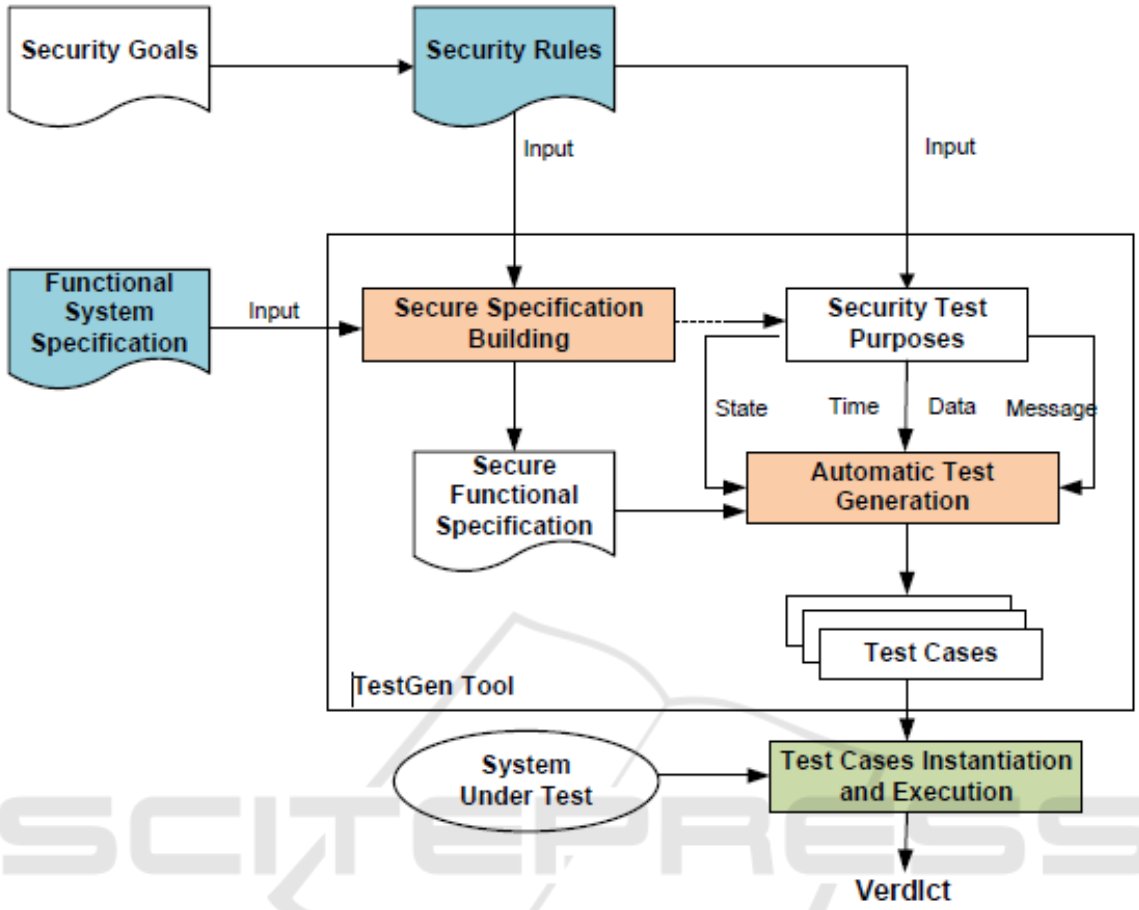

Figure 8: Our TestGen IF tool.

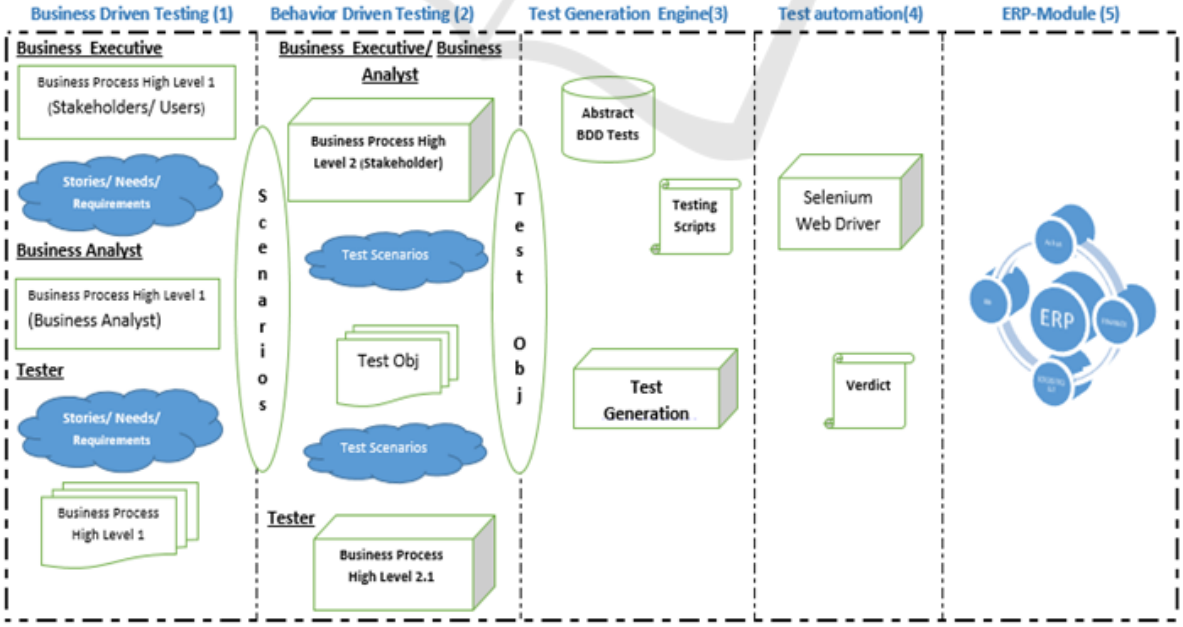

Figure 9: ERP-System Test Automation.

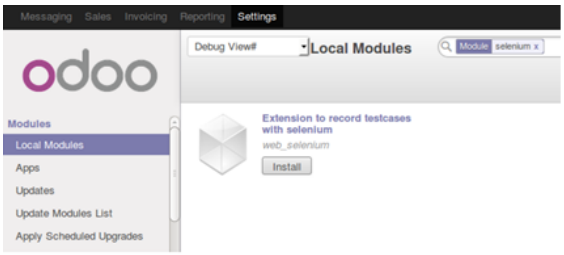

Figure 10: The Web-Selenium module.

model's name as an attribute of each related elements.

5.1 Test Automation: Workflow Sale Order in the ODOO ERP System

We automatically create a Sale Order, confirm it and deliver it as illustrated on the Figure 7.

- Install and configure ODOO;
- In ODOO, activate the modules **Sale** and **Stock**, with demo data;

5.2 Web Selenium Module Setting

Go to a directory set in the –addons-path of Odoo. In the application, go to "Settings>Users", "Edit Administrator" and tick the "Enable Technical Features" checkbox. Save and refresh the page: an "Update Modules List" entry appears in the left menu. Process with the update. You are now enable to find and install the Web-Selenium module (Fig. 10). Inspect the "Create" button from "Sales > Sales Orders List View": a data-bt-testing-model-name attribute set to "sale.order" is now set up.

5.3 Odoo Robot Framework

- Install Robot Framework first;
- Robot Framework start-up script pybot is now available;
- Setting Firefox Profile : Firefox is used as the default browser for our tests. Whenever **Pybot** executes the Open Browser keyword, Selenium creates a brand new Firefox profile;
- Get odoo-robot-framework.

Now we are ready to run the test suite (Fig. 11).

```
Documentation  Full workflow demonstration
Resource       odoo_8_0.robot

*** Test Cases ***
Valid Login
  Login

Create Sale Order
  MainMenu   Sales
  SubMenu    Sales Orders
  Button     sale.order    oe_list_add

  Many2OneSelect    sale.order    partner_id  Agrolait, Th
  Date    sale.order  date_order  08/30/2015
  Char    sale.order  client_order_ref  AGR001
  NewOne2Many    sale.order    order_line
  Many2OneSelect    sale.order.line    product_id [HDD-SH2
  Char    sale.order.line    product_uom_qty    2
  Button  sale.order.line    oe_form_button_save_and_close
  NotebookPage    Other Information
  Select-Option    sale.order picking_policy  Deliver all
  Button  sale.order     oe_form_button_save
  Capture Page Screenshot
```

Figure 11: PyBot demo-80-full-workflow.robot.

6 CONCLUSION

The paper presented an automated model-based testing approach for the test of ERP systems. A number of major activities in model-based testing are included in the experiments: modeling, test generation and test execution. A number of basic requirements are identified from the ERP specification and then described as properties in the IF language. From the basic requirements, a number of test purposes are defined and test cases are generated using the TestGen-IF tool where the generated test cases satisfy the transition coverage criterion. This approach offers several advantages:

- it covers all the steps of the software development cycle, from the business process modelling to the test generation and test execution;

- it is based on the modelling of business processes, specifying their requirements and reducing the risk of errors when compared with other models (for instance these based on the top management level);

- it provides a testing methodology that combines a set of modeling methods (high level business processes) and tools to manage the automation of ERP test generation and execution;

- it reduces time and cost of the testing procedure for ERP System.

The approach has been applied to a real case study, the ERP ODOO, and the experimental results are very promising concerning the use of model-based testing.

As future work, we plan to complete the test activity by:

- the application of monitoring techniques to analyze the behavior of the system without disrupting its operation;

- implementing an algorithm for the test architecture and measuring its complexity;

- the implementation of the improved Selenium

script.

We will perform other experiments applied to much more complex work flows. In the future applied to proprietary ERP like SAP and People soft. Also a comparative study will be carried out with similar research.

REFERENCES

(2009). Blind review. In *Formal Techniques for Distributed Systems*, pages 122–136. Springer.

Bozga, M., Graf, S., and Mounier, L. (2002). If-2.0: A validation environment for component-based real-time systems. In *International Conference on Computer Aided Verification*, pages 343–348. Springer.

Bozga M, Graf S, O. I. S. J. (2004). The if toolset, lecture notes in computer science, vol. 3185. In *2004, The IF toolset, SFM-04 (Lecture Notes in Computer Science, vol. 3185)*. Springer: Berlin, 2004; 237–267.

Builder, G. S. (2018). Github web selenium builder. https://github.com/SeleniumBuilder/selenium-builder/.

Chen, S., Elbashir, M., Peng, X., and Zhu, D. (2016). The effect of erp systems competences on business process and organizational performance. *International Journal of Management Theory and Practices*, 17(1):5–35.

Christine Paras, T. N. (2018). How to automate oracle erp testing. In *2018, How to Automate Oracle ERP Testing; Posted in Test Automation*. https://www.logigear.com/magazine/test-automation/how-to-automate-oracle-erp-testing//.

Downing, D. (2006). White paper: Managing quality in your erp project: 12 mistakes to avoid & best practices to adopt. In *2006, White paper Mentora Group*. Mentora Group, www.mentora.com.

Gerrard, P. (2007). Test methods and tools for erp implementations. In *Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION (TAICPART-MUTATION 2007)*, pages 40–46. IEEE.

Holland, C. and Light, B. (1999). A critical success factors model for erp implementation. *IEEE software*, 16(3):30–36.

Holmes, A. and Kellogg, M. (2006). Automating functional tests using selenium. In *AGILE 2006 (AGILE'06)*, pages 6–pp. IEEE.

Legge, D. (2011). White paper: Erp testing methodology approach and leading practices. In *2011, White Paper, Hitachi Consulting, www.hitachi.com*. Hitachi Consulting, www.hitachi.com.

Maag, S., Grepet, C., and Cavalli, A. (2008). A formal validation methodology for manet routing protocols based on nodes' self similarity. *Computer Communications*, 31(4):827–841.

Monier, M. and El-mahdy, M. M. (2015). Evaluation of automated web testing tools. *International Journal of Computer Applications Technology and Research*, 4(5):405–408.

Nagpal, S., Khatri, S. K., and Kapur, P. (2015). Prioritization and ranking of erp testing components. In *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO)(Trends and Future Directions)*, pages 1–6. IEEE.

Pajunen, T., Takala, T., and Katara, M. (2011). Model-based testing with a general purpose keyword-driven test automation framework. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, pages 242–251. IEEE.

Schliesser, S. (2007). An approach to erp testing using services. In *IEEE International Conference on Software-Science, Technology & Engineering (SwSTE'07)*, pages 14–21. IEEE.

Selenium, G. W. (2018). Github web selenium. https://github.com/brain-tec/web_selenium.

Software-Testing-Help (2018). The beginner's guide to erp testing (sap testing). In *2018, Last Update, The Beginner's Guide to ERP Testing (SAP Testing) – Part 1*. https://www.softwaretestinghelp.com/guide-erp-testing-sap-testing-1/.

TestingWhiz (2017). Creating Business Value with Business-Driven Test Automation. Technical report, TestingWhiz.

Wieczorek, S. and Stefanescu, A. (2010). Improving testing of enterprise systems by model-based testing on graphical user interfaces. In *2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems*, pages 352–357. IEEE.

Wieczorek, S., Stefanescu, A., and Schieferdecker, I. (2008). Test data provision for erp systems. In *Software Testing, Verification, and Validation, 2008 1st International Conference on*, pages 396–403. IEEE.