Research Journal of Information Technology

ISSN 1815-7432

Academic Journals Inc.

www.academicjournals.com

Research Journal of Information Technology 7 (1): 42-57, 2015
ISSN 1815-7432 / DOI: 10.3923/rjit.2015.42.57

An Intentional Scenario-Based Approach for Testing Software Requirements Specifications

[1]Z. Al-Khanjari, [1,2]Y. Jamoussi, [1,2]N. Kraiem, [1]A. Al-Kindi and [1]F. Al-Salti

[1]Department of Computer Science, College of Science, Sultan Qaboos University, P.O. Box 36, Al-Khoudh 123, Muscat, Oman

[2]RIADI Laboratory, National School of Computer Science ENSI, University of Manouba, Tunisia

Corresponding Author: Z. Al-Khanjari, Department of Computer Science, College of Science, Sultan Qaboos University, P.O. Box 36, Al-Khoudh 123, Muscat, Oman Tel: 0096824141482 Fax: 0096824141407

ABSTRACT

Software Requirement Specifications (SRS) provide a complete description of the software functionality, behavior and its constraints. Most bugs in software are due to incomplete or inaccurate software requirements. Testing SRS can identify the requirement conflictions and ambiguities and ensure the delivery of software which meets and satisfies the user requirements. The aim of the present study is to propose and implement an approach for testing SRS. The approach validates user requirement specifications against requirements specifications extracted from the corresponding ready software and produce a testing report. This study reports the development of the semi-automated SRS toolset that implements the proposed approach. This approach was formalized using MAP formalism which is a process model based on a non-deterministic ordering of goals and strategies. The SRS testing toolset extracts requirement specifications from the implemented software chosen to be an XML schema and compares them with SRSs from the user's perspective that are collected through a set of forms. Such toolset is expected to produce an informative validation report that indicates the degree of matching between the specifications and user requirements. To test the implemented tool, an input test case can be put which is an XML Schema sample and the corresponding user requirements specifications. The specifications for each test case are extracted manually and written out. Depending on the limitation of time, the number of test cases to be run will be three test cases; with complete entities, attributes and relationships, with some mismatches, with some mismatches but relationships.

Key words: SRS, design and development, testing database, validation, formal verification

INTRODUCTION

Every business ought to exist because there is a need for what it provides. It is the responsibility of every business to serve these needs to the best of their ability. Too many software companies failed to deliver bug-free software that satisfies the real needs of their customers. In software development area, customers usually have an abstract idea of what they want from the software (Wiegers, 2007). Software engineers are responsible for recognizing incomplete, ambiguous, or even conflicting user requirements through requirement engineering process which is considered as the most-significant phase in Software Development Life Cycle (SDLC). Software Requirement Specifications (SRS) should provide a complete description of the software specifications and its behavior and should be clearly stated. Software bugs which are due to implementation problems,

are quiet easy to be identified through the testing phase and hence resolved. However, software bugs caused by inaccurate or ambiguous requirements produce unreliable and useless software and are often cannot be identified at all (Ryser *et al*., 1998). With the advent of advanced technologies, software becomes highly demanded to manage and utilize such technologies to serve human needs and aspirations in various fields leading to the emergence of complex, sophisticated and intelligent software. Testing such software becomes imperative and more challenging, especially when it comes to test the software validity from the stakeholder point of view. Testing Software Requirements Specifications (SRSs) plays a crucial role in software testing since it determines the software usefulness and worthiness. As a consequence, an imperious need for automated toolsets on testing Software Requirement Specifications (SRSs) arises, that can validate the implemented software against the user requirements and produce an informative testing report that measures the software conformity to user requirements and hence effectively contribute in fixing this kind of software bugs (Friedman and Voas, 1995).

Testing Software Requirements Specifications (SRS) is an emerging subfield of software testing and relatively few studies have dealt with it. This study considers the literature from three perspectives: (1) General testing of SRS, (2) Testing strategies towards database and (3) Model based approaches towards the formalization of SRS.

Regarding the first perspective to generally test the SRS, researchers assured on the importance of requirement analysis and showed how best the SRS can be collected and checked (Alfeche, 1997; Kotonya and Sommerville, 1998; Pressman, 2001; Wiegers, 2003). Later, researchers mentioned about the importance of involving relevant stakeholders and the users to determine specific feature expectations, resolution of conflict or ambiguity in requirements (Stellman and Greene, 2005; Abran and Moore, 2005; Mishra *et al*., 2008; Berkovich *et al*., 2009). Pandey *et al*. (2010) emphasized that achieving an effective requirements management requires executing planning phase independently. Pavanasam *et al*. (2010) confirmed on the importance of both the requirements change management and requirements analysis phases and proposed a model for requirements inspection. Margarido *et al*. (2011) proposed to analyze the root causes of problems and build checklist to verify the software requirements specifications by classifying defect types in requirements specifications. Torkar *et al*. (2012) described how analysis can predict the impact of requirements change. Gorschek *et al*. (2012) have concluded that if SRS collected and verified then implementation costs and resources can be estimated for requirements comparison and prioritization. Khan *et al*. (2013) provided a review on software requirements issues and how to manage SRSs. Al-Khanjari (2014) discussed how to develop a good software requirements and proposed a systematic approach to test software requirements and verify them against stakeholder's vision.

Considering the second perspective testing strategies towards database, few studies have specifically dealt with testing strategies towards database applications. Spence (1994) tested ready software against user requirements specifications. At the implementation level, he used relational database schema to represent the software being tested, extract the requirement specifications and then compare them to user requirements specification collected in a set of forms. After comparison, their software produced a testing report indicating the level of matching between the two specifications. In 2005, the most related work found was proposed by Chays (2005). In his PhD dissertation, he developed a toolset that gathers required information from database schema, fills database with data, generates test cases for database application, validates the database state and assists the tester in checking the database. Another related study was the funded research

project at the Sultan Qaboos University in Oman (Al-Khanjari *et al.*, 2010). The principles of the project relied on Chays (2005) approach. It used a database schema to collect relevant information in order to generate test cases and states. A template database application has been created from the database schema to be populated with the test cases and states to finally generate a comparison based testing results.

With respect to the third perspective, the lack to correctly consider the requirements had led to use models and scenarios/use cases in testing. With model popularization in software design and development (e.g., UML), model-based testing approach gained attention. Al-Far and Whittaker (2001) introduced a model-based testing approach which tasks were associated with finite state models. These models present the requirements specifications as software behavior models. Smialek (2005) stated that Jacobson wrote in his book and articles that use cases were well suited to be used as test cases for integration testing, but did not define a procedure on how to do it. Gorschek *et al.* (2007) indicated the importance of analysis in comparing incoming requirements to the production strategies of the organization as a goal-driven model. Zagajsek *et al.* (2007) mentioned that using incomplete requirements specification and adopting improper requirements management techniques are the main reasons behind any failed project. Therefore, they proposed a requirements management process model for software based legacy systems. Cheng and Liu (2008) used view-based requirements elicitation techniques as a framework to merge different viewpoints of the stakeholders. Also Rolland (2007) in her chapter talked about conceptual modelling which can be used to abstract the specification of the required information system. She discussed the issue of using a goal-driven approach (e.g., MAP) to capture system intentionality in order to conceptualize a purposeful system. Jiang (2008) considered requirement engineering process as a research object and proposed a formal method for describing the processes. He also highlighted different tools and technologies that can be used during the requirement engineering process. Linz (2011) formalized and extended the notion of using scenarios to test a system (using regular grammars and deterministic finite-state machines, or defining scenario lifecycle diagrams). Yet despite the ubiquity of scenario approaches, a practical method supporting testers in developing test cases from scenarios has not emerged yet.

The present study aims to meet in the middle the three perspectives by using systematic test case development which is taken up early in the development process. Furthermore, synergies between the phases of system analysis and specification and system test by reuse of scenarios were also utilized. By utilizing the results of the previous researches and projects, this study also aims to implement the proposed approach and develop a testing toolset that validates a ready developed software requirements specification (e.g., XML Schema) against the collected user requirements specifications and produces a comprehensive and informative testing report that includes the testing outcomes and the results from the comparison. The testing report is expected to have a significant benefit in the case of validating the earlier parts or versions of the software when an iterative or incremental development approach is adopted.

This study proposes an approach for testing the software requirements specifications. It makes substantial and original contributions to the software engineering scholarly body of knowledge in the main research areas of software components, software component testing, model-based testing, UML-based testing and scenario-based testing. However, our aim is to come up with a well-defined and to propose a guidance centered process. Hence, we propose to formalize the approach by using MAP formalism through the use of scenarios. This approach is based on a goal-perspective which is the MAP-driven process modeling approach.

TESTING SOFTWARE REQUIREMENTS SPECIFICATIONS (SRSs)

Software Requirements Specifications (SRSs): The notion of scenarios is crucial in SRS. Scenarios are any form of description or capture of user system interaction sequences. We define the terms scenario, use case and actor as follows:

Step 1: Scenario: An ordered set of interactions between partners, usually between a system and a set of actors external to the system. May comprise a concrete sequence of interaction steps (instance scenario) or a set of possible interaction steps (type scenario)

Step 2: Use case: Smialek (2005) stated that a sequence of interactions between an actor (or actors) and a system triggered by a specific actor which produces a result for an actor. A type scenario in our terminology

Step 3: Actor: A role played by a user or an external system interacting with the system to be specified

Scenario-based testing: Scenario-based testing is a software testing activity that uses scenario tests, or simply scenarios which are based on a hypothetical story to help a person think through a complex problem or system. They can be as simple as a diagram for a testing environment or they can be a description written in prose. These tests are usually different from test cases in that test cases are single steps and scenarios cover a number of steps. Scenarios are also useful to connect to documented software requirements, especially requirements modeled with use cases. Within the Rational Unified Process, a scenario is an instantiation of a use case (take a specific path through the model, assigning specific values to each variable). More complex tests are built up by designing a test that runs through a series of use cases. Scenario testing works best for complex transactions or events, for studying end-to-end delivery of the benefits of the program, for exploring how the program will work in the hands of an experienced user and for developing more persuasive variations of bugs found using other approaches.

Intentional process modeling: Process modeling is considered today as a key issue in Software Engineering (SE). Indeed, recent interest in process modeling is changing the focus from the product to the process view in systems development. The improvement of development processes can be benefic for the enhancement of productivity and systems quality. However, recent in-depth studies of software development practices demonstrate that the knowledge about development process are very modest (Kraiem, 1997). Thus, to assure the systems processes development, there is a great need for "a conceptual process model framework" (Selmi *et al*., 2005). Process modeling is a new research area and there is no consensus about a good formalism to represent processes or even the final objectives to reach (Rolland and Prakash, 2007).

Process models may be constructed for different reasons to fulfill different purposes. One purpose may purely be descriptive which consists of recording how some processes or classes of processes are actually performed. Alternatively, models may be prescriptive, i.e., models are constructed to guide, support and provide advises or instructions to developers. Another way is to consider process models in terms of the process aspect that they address: some focus on managerial aspects of the development process whereas others have technical concerns.

In this section we first introduce the key concepts of a MAP and their relationships. Then we define MAP components as process for modeling the testing process. This process is modeled using MAP formalism which is a process representation system based on a non-deterministic ordering of goals and strategies (Selmi *et al*., 2005). A MAP can be represented as a labeled directed graph as shown in Fig. 1. The nodes represent goals and the links between nodes correspond to strategies.

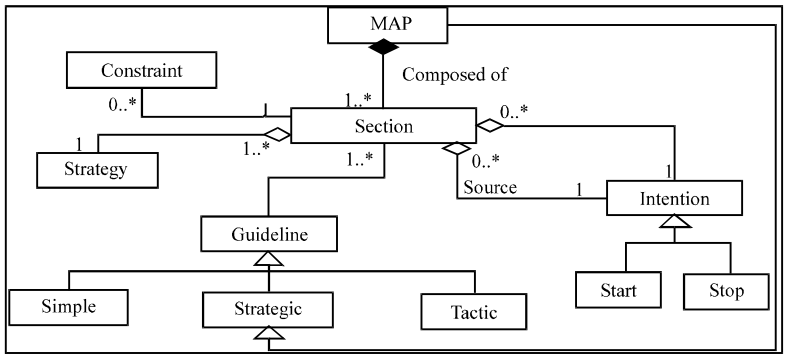

Fig. 1: MAP process meta-model

The directed nature of the graph shows the order of the different goals. A MAP is composed of one or more sections. A section is a triplet (source intention I, target intention J, strategy Sij) that captures the specific manner to achieve the intention J starting from the intention I with the strategy Sij. An intention is expressed in natural language and is composed of a verb followed by parameters. Each MAP has two special intentions "Start" and "Stop" to, respectively begin and end the navigation in the MAP. Each intention can only appear once in a given MAP. For each section a guideline is associated which is a set of indications on how to proceed to achieve an objective or perform an activity. A guideline can be: Simple, Tactic or Strategic. Three types of guidelines can be distinguished (Rolland and Prakash, 2007):

- A guideline named "Intention Achievement Guideline" (IAG) is associated to each section providing an operational mean to satisfy the target intention of the section. The IAG can be one of the aforementioned types namely tactic or simple or strategic
- "Strategy Selection Guideline" (SSG) determines which strategies connect two intentions and helps to choose the most appropriate one according to the given situation. It is applied when more than one strategy exists to satisfy a target intention from a source one. The SSG is always tactic guidelines
- "Intention Selection Guideline" (ISG) determines which intentions follow a given one and helps in the selection of one of them. It results in the selected intention and the corresponding set of either IAGs or SSGs. The former is valid when there is only one section between the source and target intentions, whereas the latter occurs when there are several sections. ISG is always tactic guidelines

MATERIALS AND METHODS

Testing software requirements specifications is a subject that requires too much studies and research, because user's requirements are closely linked to the user perception. In this subject, it is important to acquire a great bibliographical base to be able to analyze the results gotten by other researchers of the area. In the same way, in order to discover possible imperfections that can compromise the study, it is important to interview users who are directly involved with this subject. Thus, it can be detached a methodology that could be broken down into three-steps:

Step 1: Bibliographical research
Step 2: Experience relations
Step 3: Case studies

In the first step, we conducted a literature review, from three perspectives, relating to the existing approaches dealing with the problem: (1) General testing of SRS, (2) Testing strategies towards database and (3) Model based approaches towards the formalization of SRS.

Then, the interest focused on the weaknesses and limitations that can represent the existing model based approaches. Among aspects that these approaches do not allow to take into account, we can mainly cite the lack of a formalized process to help testing SRS.

In the second step, various interviews have been made with users in order to identify a couple of best practices that can be followed when conducting testing SRS. Based on these interviews, an Intentional Scenario-Based Approach have been suggested for Testing Software Requirements Specifications that aims to assist: (1) The elicitation of strategies described in an intentional manner and (2) The construction of the model representing the testing process.

In the third step, an empirical study based on a series of controlled experiments was conducted in order to assess the impact of the proposed approach. Then we discussed the benefit of the proposed approach.

Proposed approach: The method presented here comprises four main stages: Scenario creation, scenario formalization, test case derivation and intentional testing process. All stages are described in more detail as following.

Scenario creation: Most scenario processes used today are lacking a step procedure for the creation and use of scenarios. For this reason a procedure was defined to elicit requirements and document them in scenarios (Table 1). A scenario description template was used to document and format narrative scenarios. Thus adherence can be enforced to a common layout and structure.

Actors, in-and outputs and events (as determined in steps 1-3) are uniquely named. A glossary of terms including a description of all actors, in-and outputs and all events is created. The coarse scenarios created in step 5 are short natural language descriptions of the interaction and do not feature a step-by-step description yet. In step 6, instance scenarios are transformed into type scenarios and scenarios are prioritized thus allowing for release planning. Validation activities are interspersed throughout the development process (step 9 and 15). A full description of the scenario

Table 1: Scenario creation procedure

No.	Step description
1	Find all actors (roles played by persons/external systems) interacting with the system
2	Find all (relevant system external) events
3	Determine inputs, results and output of the system
4	Determine system boundaries
5	Create coarse overview scenarios (instance or type scenarios on business process or task level)
6	Prioritize scenarios according to importance, assure that the scenarios cover system functionality
7	Create a step-by-step description of events and actions for each scenario (task level)
8	Create an overview diagram and a dependency chart
9	Have users review and comment on the scenarios and diagrams
10	Extend scenarios by refining the scenario description, break down tasks to single working steps
11	Model alternative flows of actions, specify exceptions and how to react to exceptions
12	Factor out abstract scenarios (sequences of interactions appearing in more than one scenario)
13	Include non-functional (performance) requirements and qualities in scenarios
14	Revise the overview diagram and dependency chart
15	Have users check and validate the scenarios (Formal reviews)

not a linear process as it might appear from Table 1. The steps are highly intertwined and activities creation method can be found in the study of Smialek (2005). The scenario creation procedure is are performed in parallel and iteratively.

Scenario formalization: Sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in which order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. A sequence diagram was created using many software but for example we have taken the Rational Rose since it is an IBM product. UML Model file will be saved with an extension MDL. A sequence diagram shows as parallel vertical lines (lifelines), different processes or objects that live simultaneously and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner as illustrated in Fig. 2.

Test case derivation: For generating the test case from the sequence diagram, we first extract the necessary information from the diagram. For that write parser in java to extract all possible information from file. Based on the extracted information, a Sequence Dependency Table (SDT) is generated as illustrated in Table 2. With the help of SDT a test path are generated, by applying a semi-automatic process that prioritize the above mentioned test cases.

Intentional testing process: The primary research motivation of this study is to address the four most important challenging research problems described in previous section. For that reason, a SRS and intentional testing evaluation process was introduced, with the ultimate goal to produce trusted quality software. In the following the stages of the proposed approach was described.

After finalizing the scenario which encapsulates the software requirements specifications, a semi-automatic process is used to semi-convert the scenario into a schema file. The testing process begins by extracting required information from the schema that contains the specifications of the

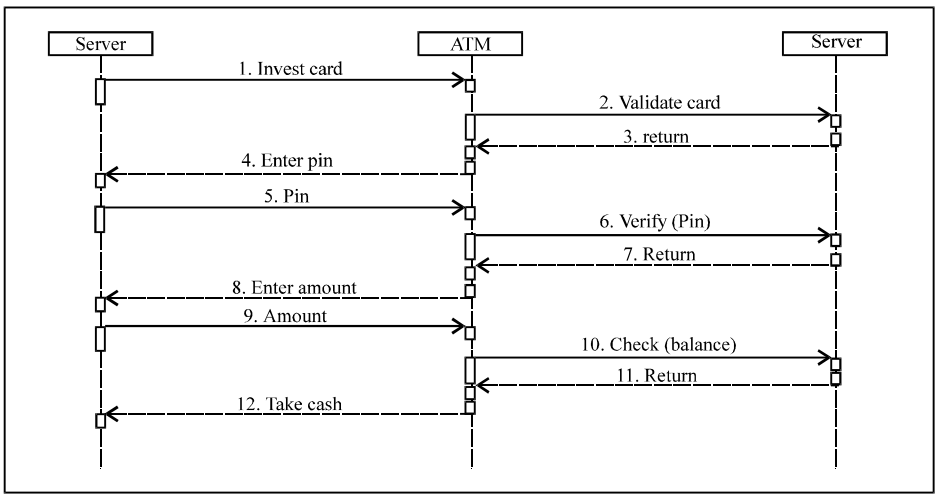

Fig. 2: Sequence diagram

software being tested. On the other hand, a set of forms are used to collect the software requirements specifications from the stakeholder's point of view. The information extracted from the schema (i.e., the requirements specifications of the implemented software) and those collected from the users are compared and a testing report is generated. The procedure steps are discussed below. Figure 3 summarizes the testing methodology.

- **Extract information from the schema:** The user is allowed to upload the XML schema into the system in order to be validated. The XML schema should be in a specific format. The obtaining of the schema is done by using a tool. The SRS testing toolset, takes this schema

Table 2: Sequence dependency table

Symbol	Activity name	Sequence No.	Dependency	Input	Expected output
A	Insert card	1			
B	Validate (card)	2	A	Card	True (enter the pin) False (End)
C	Enter pin	3	B	Pin	Pin
D	Pin	4	C	User types the pin	Receives the pin and send it to the server
E	Verify (pin)	5	D	Pin	True (enter amount) False (End)
F	Enter amount	6	E	Amount	Amount
G	Amount	7	F	User types the amount	Receives the amount and send it to the server
H	Check balance (amount)	8	G	Amount	True (take cash) False (End)
I	Take cash	9	H	Cash	End
J	End	10	I		

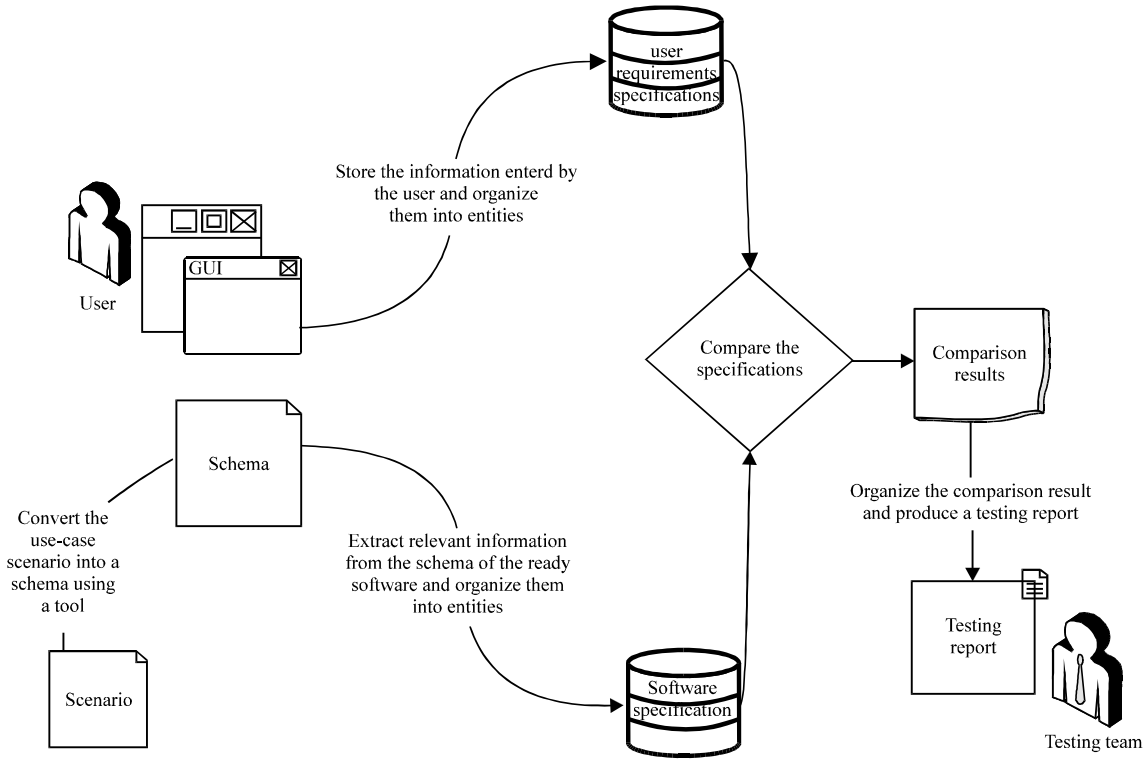

Fig. 3: SRS testing methodology

and extracts the relevant information (i.e., tables, attributes and their data types, primary keys and relationships between tables), stores the extracted information as objects of classes created to facilitate the comparison process. This intention is achieved by the organizing information into entities strategy

- **Collect user requirement specifications:** A set of forms displayed to enable the user to enter the requirements specifications. The user is able to specify names for the tables. She/he is also able to add, delete, or modify attributes for each table (name, data type, primary key or not). In addition, relationships between tables can be added, modified, or removed. This information is then structured using the same strategy and as similar objects as those created in step 1. The reason behind structuring the information into similar objects for simplifying the comparison process
- **Compare the objects created in step 2:** The comparison is done by the strategy of comparing entities created in step 1 and 2. The comparison process includes comparing the number of tables entered by the user with the number of tables extracted from the XML schema. The attributes and their data types in each table from both specifications are compared. The primary key and the relationships between tables are also compared. Notice that the names of the tables and the attributes are not compared because of time limitation, so we assume that they are the same
- **Generate a testing report:** The final step is to generate an informative report. The report is created by organizing comparison results. The report informs the user about the result of testing her/his software. It indicates the percentage of matching between the specifications required by the user and the specifications extracted from the schema. The user will have an option of saving the report in a text file or just displaying it on the screen

The SRS testing methodology was proposed to represent with MAP formalism. The present model process is composed of intentions to achieve and strategies. Our process begins with the "Start" intention and finishes with the "Stop" intention. The intention "Convert scenario into schema" is achieved by using a tool. The result of this achievement is extracting relevant information from schema by organizing information into entities. The intention "Store information entered by user" is achieved by organizing them into entities. To achieve the comparison of the specification, different entities were compared. The organization of comparison result is the strategy used to produce testing report. The process is finished when we reach the intention "Stop". Figure 4 shows the SRS testing methodology model process.

The intention "Convert scenario into schema" can be refined as shown in Fig. 5 into another MAP which has as purpose to explain and provide more details to achieve this intention. Two strategies can be chosen to generate entities from a schema which are the automatic reverse process and the semi-automatic one.

SRS testing toolset: The main objectives behind developing SRS testing toolset are:

- Check whether the implemented software meets the user requirements specifications, i.e., validate the implemented software
- Use the implemented database application (schema) to determine and specify the set of requirements to be tested and validate them against the collected user requirements
- Produce interactive and friendly forms to collect user requirements, based on the requirements extracted from the implemented database schema

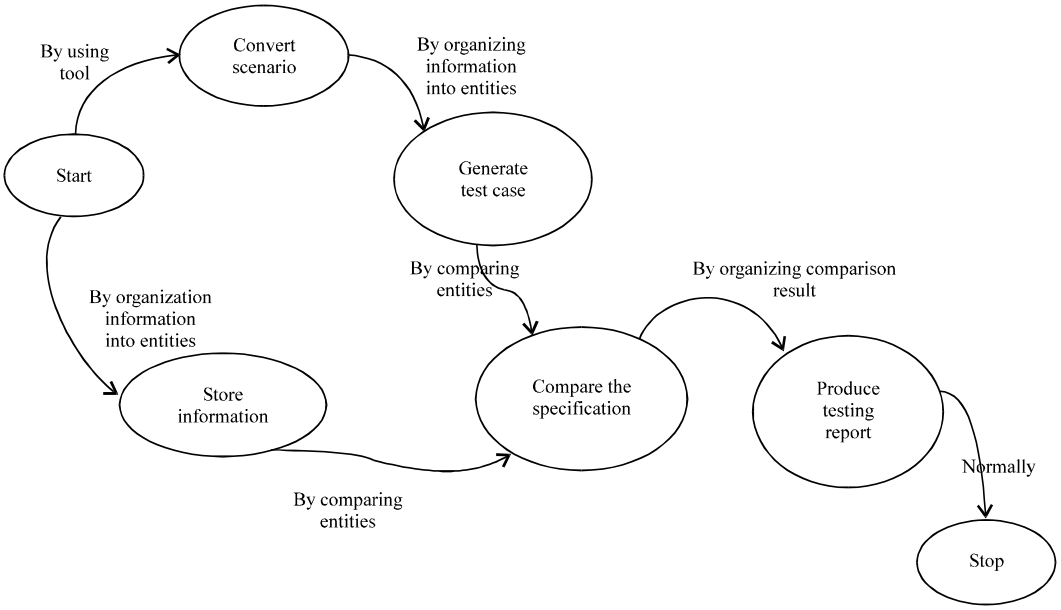

Fig. 4: SRS testing methodology model process

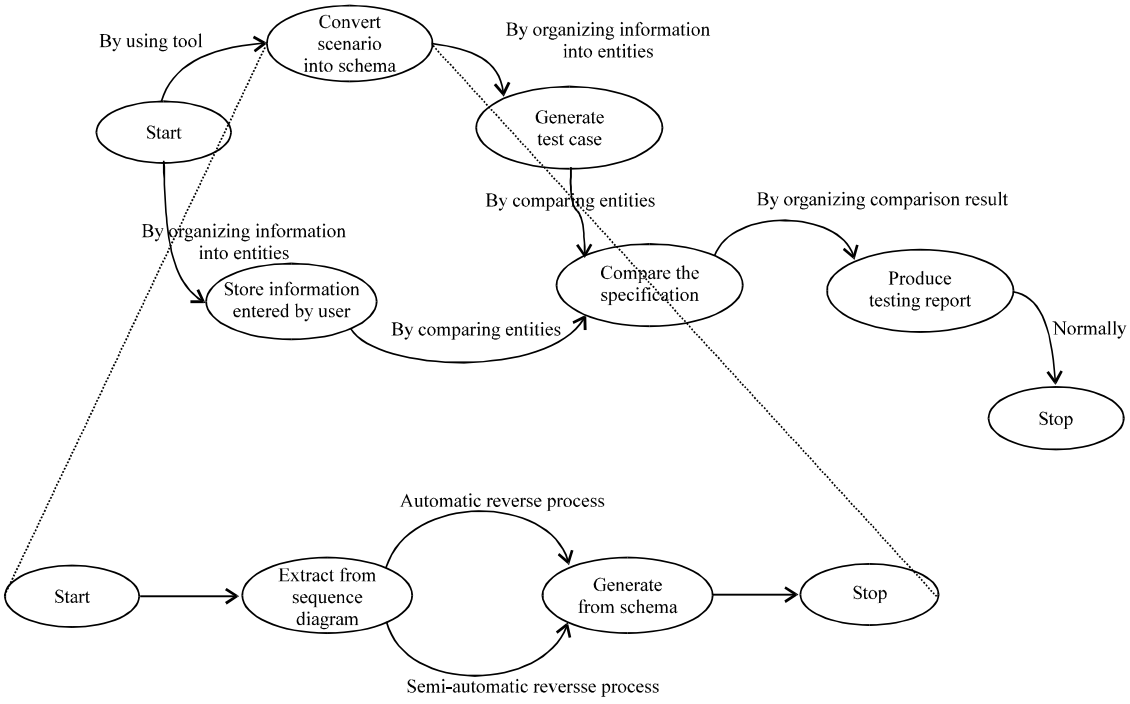

Fig. 5: Refinement model process of "convert scenario into schema"

- Design data validation for user input using a two differing approaches: User interface code and application code (Mclaughlin, 2000). This mechanism coordinates with the user during collecting his/her requirements by designing forms that exploits the data validation feature provided by the GUI controls. Such a feature allows applying validation rules on data entry in order to check correctness, meaningfulness and security of data that are input to the system

For example:

- Control data entry formats with input masks which define how users must enter data in specific fields to help maintain consistency. For example, an input mask is set for an input field so that users can only enter telephone numbers in the Omani format or addresses in the USA format
- List of possible values are displayed to the user and he/she is imposed to select one of them
- Include some symbols indicating the constraints on some input fields (e.g., use asterisk '*' to indicate required fields) and provide clear captions and legends that interpret the appearing terms and facilitate the data entry process

Architecture: The toolset validates requirements specifications extracted from an XML Schema against user requirements and produce a testing report that measures the software conformity to user requirements. The toolset is composed of four main tools:

- **Schema parser tool:** Extracts the software specifications from the schema
- **User requirements specification collector tool:** Collects user requirements specifications through a Graphical User Interface (GUI)
- **Comparison tool:** Compares the collected specifications and produces comparison results
- **Testing report generator tool:** Produces a testing report based on the comparison results

Figure 6 shows an abstract architecture of SRS testing toolset.

Implementation: Testing software requirements specifications is crucial to software validation, since it determines the software usefulness and adequacy. SRS testing toolset was developed as a primitive automated testing toolset for validating XML Schema in XML-based database systems. The main purpose is to validate the conformity of implemented software which is in this case an XML Schema, against its user requirements. The toolset is composed of four main tools:

- **XML schema parser tool:** The XML parser was designed to take an XML Schema file to extract XML Schema specifications (i.e., software specifications). Using the JDOM library (Hunter and McLaughlin, 2011), the XML Schema file will be stored as a JDOM Document to facilitate the parsing process. The extracted Schema specifications are organized into a set of entities that represents the software specifications

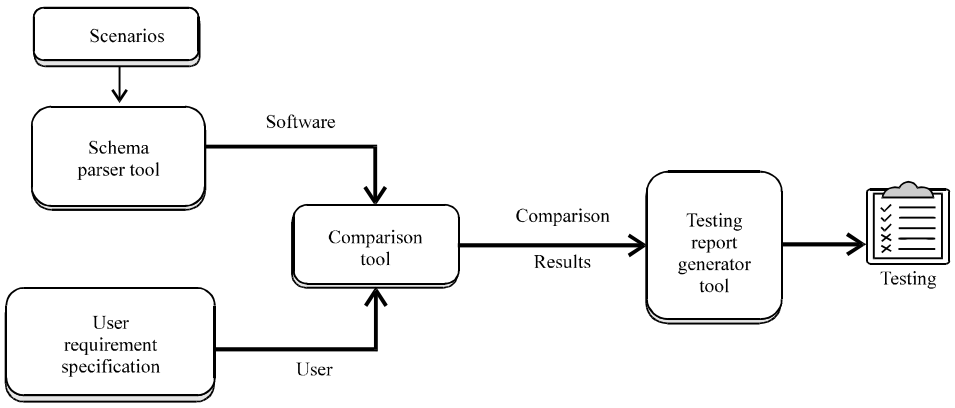

Fig. 6: An abstract architecture of SRS testing toolset

- **User requirements specification collector tool:** This tool is composed of a set of user-friendly forms that are used to collect user requirements specifications and then organize these specifications into a set of entities in order to be prepared for the comparison process
- **Comparison tool:** Compares the collected specifications and produces comparison results. The comparison process starts by comparing the user specifications entities against those extracted from the XML Schema. The comparison results are produced
- **Testing report generator tool:** After the comparison process is completed, the user can get a testing report based on the comparison results. This tool organizes the comparison results into a comprehensive and readable testing report. The testing reports includes information about the matched and mismatched entities between the specifications, entities, matched and mismatched attributes for each matched entities, matched and mismatched relationships in the specifications. The user can display or save a copy of the testing report

The SRS testing toolset is designed as a component-based system, where each tool was developed as a stand-alone component according to JavaBeans™ API specification (Oracle, 2011). JavaBeans APIs define a software component model in Java, so that these components can be composed together into applications by developers. The components (i.e., tools) communicate with each other directly by invoking methods. Developing the SRS testing toolset as component-based system took the advantages of such architecture, such as:

- The most important feature of a component is the separation of its interface from its implementation. The services provided by the component are only visible through its interface
- There exists a strong relation between object-oriented programming OOP and components. Component models (also called component standards) COM/DCOM, .NET, Enterprise Java Beans (EJB) and CORBA Component Model (CCM) relate Component Interface to Class Interface (Oracle, 2011; Harold, 2002)
- Components adopt object principles of unification of functions and data encapsulation. All system processes are placed into separate components so that all of the data and functions inside each component are semantically related (just as with the contents of classes). Because of this principle, it is often said that components are modular and cohesive
- While researchers in academia define components as well defined entities (often small and with easily understood functional and non-functional features), industry sees components as parts of a system which can be reused. Such components (or rather reusable entities) are of extreme importance as the larger the components, the greater the productivity which can be achieved by their reuse (Linz, 2011)

The main reasons behind the selection of XML Schema to represent the software system being tested are: (1) XML Schema uses basic XML syntax, (2) Supports namespace recommendations, (3) Enables the definition of vocabularies that utilize namespace declarations and (4) Allows the validation of text element content based on built-in and user-defined data types (Mclaughlin, 2000; Oracle, 2011).

The toolset was implemented using Java programming language which provides powerful Application Programming Interfaces (APIs) for building reliable applications and user-friendly GUIs. The JDOM library was used to facilitate the development of the XML parser tool. JDOM Library is an open source library that integrates with Document Object Model (DOM) and Simple API for XML (SAX), supports XPath and XSLT. It enables the XML Schema file to be stored as a document object which then simplifies the access of information. The drawback of wasting large

amounts of storage when converting the files to JDOM objects does not have a significant impact on the toolset performance because most of the time the size of XML Schema files is not very large (Hunter and McLaughlin, 2011; Oracle, 2011).

RESULTS AND DISCUSSION

The main objective of the developed SRS testing toolset is to validate the ready developed software requirements specifications against the collected user requirements specifications. There were several attempts to solve such a problem (Chays, 2005; Abran and Moore, 2005), but the differences were mainly in the overall approached methodology. The most existing approaches consider the testing SRS regardless the user's viewpoint and hence they are considered rigid. The new proposed approach overcomes this lack by proposing a multi-process and contextual-oriented modelling which provides to the tester more flexibility and gives him ability to personalize his testing process.

SRS testing toolset extracts requirement specifications from the implemented software chosen to be an XML schema and compares them with SRSs from the user's perspective that are collected through a set of forms. But, the system could not use existing XML parsing tools, because all XML parsers that found validate XML documents that contain actual data against XML Schema files. The SRS testing toolset developed an XML parser according to the system requirements. The XML schema parser is designed to extract information about the database from the schema and organize them into suitable data structures (Objects) in order to be ready for comparison with the user requirements. There were two reasons for using such approach in developing the XML parser:

- We always used to provide the database schema, instead of the actual data, to the testing team due to privacy and security issues
- Existing XML parsers are designed to validate given XML files against XML schema. Such approach requires ready XML files with data in order to use the XML parser
- "Collecting user requirements" forms are basically designed to coordinate the user during collecting his/her requirements. The design forces the user to fill the data according to a specific manner

Our approach presents a development of the semi-automated toolset SRS. It is based on a process model based on a non-deterministic ordering of goals and strategies. SRS testing toolset extracts requirement specifications from the implemented software chosen to be an XML schema and compares them with SRSs from the user's perspective that is collected through a set of forms. Furthermore, this proposed approach covers transparently the different testing approaches by adding more flexibility and by focusing much more in the situational aspect.

CONCLUSION

Today many varieties of business software are available and used by different organizations, where each software achieves different work. Software is considered an important asset to the business and it must conform to the business requirements. Software bugs caused by inaccurate or ambiguous requirements produce unreliable and useless software and are often cannot be identified at all. As a consequence, an imperative need for tools on testing Software Requirement Specifications (SRS) arises, that can validate the implemented software against the user requirements and produce an informative testing report that can effectively contribute in fixing this kind of software bugs. This study proposed and implemented a scenario-based approach for testing

Software Requirements Specifications (SRS). The testing begins by creating a scenario that encapsulates the requirements specifications of the software being tested which in its turn is converted to a schema using a special tool. Then, test cases for the complete system are obtained from sequence diagrams. After that, a semi-automatic process is performed in order to validate the requirements in the software to be tested. Finally, a test report summarizing the previous steps is generated. An implemented toolset, called SRS testing toolset, is developed for validating user requirement specifications against requirements specifications extracted from the corresponding ready software. The main components or tools of this toolset are: (1) A tool for extracting requirement specifications from the implemented software (e.g., schema), (2) A set of forms to collect software requirement specifications from the user's perspective, (3) A tool that compares the extracted software specifications and the collected user requirements specifications and (4) A tool that produces the testing report that will indicate whether the implemented software satisfies all user requirements or not, by determining the compatibility degree between the specifications.

Further investigations are required to efficiently overcome the various aspects of incompetency of the current testing toolset, such as:

- Perform more validation tests on the developed testing toolset in order to discover significant bugs and guarantee the program reliability
- Develop the XML Schema parser to be more general (i.e., able to parse different formats of XML Schema)
- Improve the analysis of the comparison results in order to produce more informative testing report
- Improve the GUI of the toolset to monitor the user input and detect logical errors
- Software engineers regard the Component-Based Architecture (CBA) as part of the starting platform for Service-Oriented Architecture (SOA) and Event-Driven Architecture (EDA). For example, in Service-Oriented Architectures (SOA), a component is converted by the web service into a service and subsequently inherits further characteristics beyond that of an ordinary component. In addition, components can produce or consume events and can be used for Event-Driven Architectures (EDA). Therefore, the system can be observed as service-oriented system, where each component (i.e., tool) is converted by the web service into a service and subsequently inherits further characteristics beyond that of an ordinary component

REFERENCES

Abran, A. and J. Moore, 2005. Software Requirements. John Wiley and Sons, New York.

Al-Far, I. and J. Whittaker, 2001. Model-Based Software Testing. In: Encyclopedia on Software Engineering, Marciniak, J. (Ed.). John Willey and Sons, New York.

Al-Khanjari, Z.A.S., S. Kutti, A. Al-Hamadani and A. Al-Hosni, 2010. Testing relational database application software. Department of Computer Science, College of Science, Sultan Qaboos University.

Al-Khanjari, Z.A.S., 2014. Proposing a systematic approach to verify software requirements. J. Software Eng. Applic., 7: 218-224.

Alfeche, R., 1997. Requirements Engineering: A Good Practice Guide. John Wiley and Sons, Hoboken.

Berkovich, M., J.M. Leimeister and H. Krcmar, 2009. Suitability of product development methods for hybrid products as bundles of classic products, software and service elements. Proceedings of the ASME International Design Engineering Technical and Computers and Information in Engineering Conference, August 30-September 2, 2009, San Diego, California, USA.

Chays, D., 2005. Test data generation for relational database applications. Ph.D. Thesis, Polytechnic University.

Cheng, J. and Q. Liu, 2008. Using stakeholder analysis for improving statechart merging in software requirement management. Proceedings of the 9th International Conference for Young Computer Scientists, November 18-21, 2008, Hunan, pp: 1217-1222.

Friedman, M. and J. Voas, 1995. Software Assessment: Reliability, Safety, Testability. Wiley, New York, ISBN: 9780471010098, Pages: 271.

Gorschek, T., P. Garre, S.B.M. Larsson and C. Wohlin, 2007. Industry evaluation of the requirements abstraction model. Requirements Eng. J., 12: 163-190.

Gorschek, T., A. Gomes, A. Pettersson and R. Torkar, 2012. Introduction of a process maturity model for market-driven product management and requirements engineering. J. Software Evol. Process, 24: 83-113.

Harold, E.R., 2002. Processing XML with Java: A Guide to SAX, DOM, JDOM, JAXP and TrAX. 1st Edn., Addison-Wesley Professional, Boston, MA., ISBN-13: 078-5342771862, Pages: 1120.

Hunter, J. and B. McLaughlin, 2011. JDOM v1.1.3 API specification. http://www.jdom.org/docs/apidocs.1.1/index-files/index-15.html.

Jiang, X., 2008. Modeling and application of requirements engineering process metamodel. Proceedings of the IEEE International Symposium on Knowledge Acquisition and Modeling Workshop, December 21-22, 2008, Wuhan, pp: 998-1001.

Khan, M.N.A., M. Khalid and Sami ul Haq, 2013. Review of requirements management issues in software development. Int. J. Modern Educ. Comput. Sci., 5: 21-27.

Kotonya, G. and I. Sommerville, 1998. RE, Processes and Techniques. 1st Edn., John Wiley and Sons Ltd., New York, ISBN 13: 9780471972082, Pages: 282.

Kraiem, N., 1997. Use of the process Meta-model to describe requirements engineering methodologies. Proceedings of the 3rd IEEE International Conference on Engineering of Complex Computer Systems, September 8-12, 1997, Como, Italy, pp: 306-318.

Linz, P., 2011. An Introduction to Formal Languages and Automata. 5th Edn., Jones and Bartlett Publishers, Inc., Burlington, ISBN: 9781449615529, Pages: 437.

Margarido, I.L., J.P. Faria, R.M. Vidal and M. Vieira, 2011. Classification of defect types in requirements specifications: Literature review, proposal and assessment. Proceedings of the 6th Iberian Conference on Information Systems and Technologies, June 15-18, 2011, Chaves, pp: 1-6.

Mclaughlin, B., 2000. Validation with Java and XML schema. JavaWorld, Inc., September 8, 2000.http://www.javaworld.com/article/2076170/java-se/validation-with-java-and-xml-schema-part-1.html

Mishra, D., A. Mishra and A. Yazici, 2008. Successful requirement elicitation by combining requirement engineering techniques. Proceedings of the International Conference on the Applications of Digital Information and Web Technologies, August 4-6, 2008, Ostrava, pp: 258-263.

Oracle, 2011. Java™ platform, standard Edition 7, API specification-Package java.beans. http://docs.oracle.com/javase/7/docs/api/java/beans/package-summary.html#package_descriptio

Pandey, D., U. Suman and A.K. Ramani, 2010. An effective requirement engineering process model for software development and requirements management. Proceedings of the International Conference on Advances in Recent Technologies in Communication and Computing, October 16-17, 2010, Kottayam, pp: 287-291.

Pavanasam, V., C. Subramaniam, T. Srinivasan and D.J.K. Jain, 2010. Membrane computing model for software requirement engineering. Proceedings of the 2nd International Conference on Computer and Network Technology, April 23-25, 2010, Bangkok, pp: 487-491.

Pressman, R.S., 2001. Software Engineering: A Practitioner's Approach. 5th Edn., McGraw-Hill, USA.

Rolland, C., 2007. Capturing System Intentionality with MAPs. In: Conceptual Modelling in Information Systems Engineering, Krogstie, J., A.L. Opdahl and S. Brinkkemper (Eds.). Springer, Berlin, Heidelberg, pp: 141-158.

Rolland, C. and N. Prakash, 2007. On the adequate modeling of business process families. Proceedings of the 8th Workshop on Business Process Modeling, Development and Support, June 11-12, 2007, Norway, pp: 1-8.

Ryser, J., S. Berner and M. Glinz, 1998. On the State of the art in requirements-based validation and test of software. Technical Report of the State of the Art in Requirements-based Validation and Test of Software, University of Zurich.

Selmi, S.S., N. Kraiem and H.B. Ghazala, 2005. WApDM: A multi-process approach for web applications design. Proceedings of the International Conference on Internet Computing, June 27-30, 2005, Las Vegas, pp: 115-125.

Smialek, M., 2005. Accommodating informality with necessary precision in use case scenarios. J. Object Technol., 4: 59-67.

Spence, C., 1994. Generation of software tests from specifications. Proceedings of the 2nd Conference on Software Quality Management, July 26-28, 1994, Edinburgh, Scotland, UK.

Stellman, A. and J. Greene, 2005. Applied software project management. O'REILLY, December 6, 2005.

Torkar, R., T. Gorschek, R. Feldt, M. Svahnberg, U.A. Raja and K. Kamran, 2012. Requirements traceability: A systematic review and industry case study. Int. J. Software Eng. Knowledge Eng., Vol. 22. 10.1142/S021819401250009X

Wiegers, K.E., 2003. Software Requirements. 2nd Edn., Microsoft Press, Washington.

Wiegers, K.E., 2007. Ambiguous software requirements lead to confusion, extra work. http://searchsoftwarequality.techtarget.com/tip/Ambiguous-software-requirements-lead-to-confusion-extra-work.

Zagajsek, B., K. Separovic and Z. Car, 2007. Requirements management process model for software development based on legacy system functionalities. Proceedings of the 9th International Conference on Telecommunications, June 13-15, 2007, Zagreb, pp: 115-122.