

CONTEXT-BASED REQUIREMENTS REPRESENTATION FOR SOFTWARE TESTING (SUMMARY OF PHD WORK IN 2011)

János OLÁH

Advisor: István MAJZIK

I. Introduction

The quality and the success of software systems depends on how well it meets its intended needs of the stakeholders. Requirements engineering (RE) is a process which involves the understanding the needs of stakeholders; understanding the context in which the system will operate; capturing the requirements into an adequate format; and validating that the documented requirements match the negotiated requirements.

Beyond these core activities, many software engineering activities and artefacts are based on the requirements, e.g., software testing. In this paper we introduce a novel requirements description method, which is able to capture requirements related to the context of software (i.e., requirements, that can be expressed solely in terms of the context), and can effectively support the generation of functional test cases.

II. Requirements engineering

By definition, requirement is “a condition or capability needed by a user to solve a problem or achieve an objective” (ISO/IEC/IEEE 24765:2010; Systems and Software Engineering). While other software engineering activities result in artefacts directly affecting the software’s behaviour (i.e., they are part of the solution), requirements engineering rather defines the problem that has to be solved. This means that functional requirement descriptions are written entirely in terms of the environment of the desired software.

In RE related research papers, RE research is decomposed into five tasks: elicitation, modelling, requirements analysis, verification and validation, and requirement management [1]. These RE activities are usually iterative, since they involve many actors (stakeholders, architects, developers, testers, etc.) with different background and focus.

These activities cannot be executed and understood separately. In this paper we want to emphasize how an adequate requirements description can help generating sound functional test cases, thus we will focus on requirements elicitation and modelling.

Requirements elicitation involves activities to understand the goals and motives of building the desired software system. Activities of elicitation comprise discussion with the stakeholders, hence informal and intuitive models are especially popular, like use-cases, scenarios (simple text, cartoons or video), demos and simulations. These promise easy comprehension and quick feedback by non-professional stakeholders.

During requirements modelling phase, the high-level models constructed for stakeholders are refactored to more detailed, precise, often formal models, that models can be used by software architects and developers to plan and create the software system. Scenario-based models are usually constructed during this phase of RE. These models contain a “step-by-step description of series of events, that may occur concurrently or sequentially” (IEEE 1362-1998 (R2007)). Scenario-based models are well-known examples of relationship between RE and software testing. For example UML sequence diagrams are popular to model the flow of messages, events and actions between the components of a system (in addition, they are still relatively easy to understand by non-technical stakeholders). Scenario-based

models contains all necessary information (i.e., input data and conditions, and expected output data and conditions), thus testers can reuse these scenario descriptions to derive test cases.

III. Context modelling

In this paper we propose a novel requirements representation approach, which utilizes the model of the software's context. Contextual RE techniques analyse stakeholders' requirements with respect to a particular context. The context is the environment in which the desired software will operate. The importance of context during requirements elicitation has long been recognized by RE research community, however in many cases the contextual techniques often focus on improving the elicitation phase (e.g., stakeholder interview design) by understanding the context [2].

In contrast to this approach, we refer to context modelling as the main part of the elicitation phase. In most cases, there are assumptions about the context, and in many cases this context is finite and entirely known. During the modelling of the context we create a metamodel that contains every possible object that may appear in the environment of the software, and every action through which the environment can affect the software.

We would like to express the functional requirements in terms of this metamodel. This means that the requirements are model instances conforming to this metamodel. Again, this representation is only appropriate in case of context-related functional requirements, where the user needs can be described entirely with objects and actions from the software's environment.

For example in case of an autonomous robot vacuum cleaner, the environment is a household. The stakeholders have the apriori knowledge of every possible object that may appear in the environment of the robot (e.g., furniture, living beings), and the possible actions are known as well (e.g., perceptions/actuators of the robot). Another example is a software with web-based interface, with given GUI objects (buttons, text boxes, combo-boxes, drop-down lists, etc.) and standard actions (e.g., click, double-click, draw, type, etc.) that can be executed on these objects.

In order to thoroughly express the software's functionalities with model instances, we need to capture the expected output as well. This means we need to complement the metamodel or create another metamodel with actions and objects, which define the output of the software. It is important to note that in many cases the input and output metamodels may be the same or at least the objects may be similar, and only the actions are different. For example in the household environment of the autonomous vacuum cleaner the objects are common for both metamodels, but the actions may be different for the output metamodel (e.g., start draining, give alert). In the other example, the output actions will contain elements associated with the server side action (e.g., page loaded, exception thrown).

Using this approach to RE elicitation, we express the software functionalities as ordered pairs of input and output models.

A. *Motivation of context-based requirements representation*

According to the context representation described above, during the elicitation phase we create a structured view of the software's environment that contains every relevant object, action and relationship among these, instead of an ad-hoc representation of context elements in different requirements without capturing their hierarchy and relations. In addition we may assign arbitrary attributes to these model elements (e.g., timing to actions, position to objects).

Beyond the requirements, using the domain-specific, structured metamodel of the context enables the representation of the necessary constraints, that for example require the presence of a certain object, or define cardinality restrictions.

The primary aim of the proposed representation is to facilitate test case derivation. We represent functional test data as model instances of the context metamodel that contain the requirements input model instances, and test executions are evaluated by searching for the output model instances. We

think that the proposed representation is advantageous in the following cases:

- To support automated test case derivation with instantiation of elements of the metamodel (i.e., model instance generation) and fulfilment of the constraints.
- To derive robustness test cases through the violation of constraints.
- To systematically combine requirements (i.e., input models of different requirements) to derive more complex test cases.
- To detect incompleteness and explore inconsistency among the requirements on the basis of the metamodel.
- To define precise coverage metrics based on the hierarchy of elements and their relations.

B. Example

Let us consider the example of a web-based software. On the login screen there are two text boxes (e.g., login name and password) and one button (e.g., login). Thus the metamodel in this simple case contains two text boxes and a button, and a two actions, namely type and click. Additionally there are actions executed by the software, for example “page load”. This simple metamodel describes the context of the program (i.e., in this case we composed one single metamodel).

A basic requirement to express may be the following: when the user provides the login credentials and then click on login button, the welcome screen is loaded.

In order to describe this requirement, we instantiate two text boxes and two type actions from the input metamodel, and associate each text box instance with two type action instance. The instances can have parameters, for example text that have to be typed, timing relations, etc. Then we create an instance of the button and an associated click action. This model instance conforming to the input metamodel will be the input model of the requirement, expressing that the user types the login credentials and then clicks on login button.

We may express the expected outcome by instantiating a page load action (i.e., the output model), which again can hold the expected page title as parameter. Using our proposed representation, this pair of input and output model will express the requirement mentioned above.

For the example above, a straightforward action metamodel can be constructed using Selenium command set. Selenium is a browser automation tool, that is very popular for web application’s test automation [3]. The command set (usually referred to as “selenese”) contains actions in three category.

Actions provide functionality to manipulate the application, e.g., click on or select an item.

Accessors are used to identify and save the current state of the application.

Assertions can verify that the state of the application matches what expected, e.g., verify that a check box is checked, etc.

In case of a web-based software, using these as actions in the metamodel and every GUI element from the interface, we get a metamodel that is appropriate to describe almost every interactions (including timing information and any desired parameter) between the software and its environment.

IV. Use of context-related requirements for software testing

Software testing is the process of evaluating the quality of the software under test (SUT) by controlled execution, usually with the primary aim to reveal inadequate behaviour. In functional (i.e., black-box) software testing we treat software as it would be a function transforming the input data to the output data. We do not have or do not use the information about the internal structure of the SUT. This means that the tester has to be familiar with relations between the SUT and the environment. At this point we may exploit the proposed requirements representation approach, since it expresses how the SUT

interact with its environment without dealing with details or internal information. These can be used to derive test case descriptions for the SUT.

A possible way of test case generation based on the requirement models is model generation, i.e., model instance creation conforming to the given metamodel. In simple cases the input model of the requirements model can be used as an input test data directly, and we can compare the output model to the actual output data. However, when we need more complicated models to exercise the SUT (e.g., we have constraints that force the presence of particular elements), we may expand the initial input model by instantiating further elements from the metamodel, and get a model that contains other objects and actions.

In the previous example, we may instantiate every clickable GUI element from the metamodel and add a click event to all of these, and then type the login credentials and click on login button. With this extension we can verify that the requirement is satisfied even if the user previously clicked every element on the web interface.

In addition, we may automate the generation of such models. Search-based software engineering (SBSE) is the use of search-based optimization algorithms (usually metaheuristic search techniques) to software engineering problems. The key ingredients for the application of search-based optimization is the choice of representation of the solutions and the definition of the fitness function. In case of test generation on the basis of context related requirements, we can represent solutions with model instances, and we can compose the fitness function from the test goals and coverage metrics (e.g., minimum number of elements, instantiation of particular elements), that will guide the model generation. Thus we can use SBSE to search for model instances that are appropriate according to our selected conditions. A search-based model generation approach is presented in [4] and [5].

V. Conclusion and future work

In this paper we have briefly introduced a novel approach for high-level description of context-related functional requirements. We create a metamodel, that describes the context of the software, and contains every possible object that may appear in the context and every possible action that may affect the software or can be executed by the software. We use the context metamodel and define requirements as model instances conforming to this metamodel. We think that this new approach to requirement modelling is advantageous when generating functional test cases.

In the future we plan to implement a framework which is able to generate test cases based on the context-related requirement models and additional boundary conditions.

Acknowledgement

This work was partially funded by TÁMOP-4.2.1/B-09/1/KMR-2010-0002 and ARTEMIS-JU Grant Agreement 100233 (R3-COP).

References

- [1] B. H. C. Cheng and J. M. Atlee, “Research directions in requirements engineering,” in *2007 Future of Software Engineering, FOSE '07*, pp. 285–303, Washington, DC, USA, 2007. IEEE Computer Society, URL: <http://dx.doi.org/10.1109/FOSE.2007.17>.
- [2] T. Cohene and S. Easterbrook, “Contextual risk analysis for interview design,” in *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, pp. 95–104, 2005.
- [3] Selenium Project, *Selenium Documentation*, 2011, URL: <http://seleniumhq.org/docs/>.
- [4] Z. Szatmári, J. Oláh, and I. Majzik, “Ontology-based test data generation using metaheuristics,” in *Proc. of the 8th International Conference on Informatics in Control, Automation and Robotics*, 2011.
- [5] J. Oláh and I. Majzik, “Search-based functional test data generation using data metamodel,” in *Proc. of the 3rd International Symposium on Search Based Software Engineering*, 2011.