Test Scenarios Generation Based on Use Cases

Fabio Roberto Octaviano, André Di Thomazzo, Kamilla Camargo, Sandra Fabbri

Federal University of São Carlos – UFSCar - São Carlos, Brazil

{foctaviano, andredt}@ifsp.edu.br,
{sfabbri, kamilla_camargo}@dc.ufscar.br

Abstract. This research group has been working towards building an environment which helps software development, providing support to some activities based on use case model. Prior to our enrollment in this research, the environment supported use case construction, requirements document analysis, requirements traceability and use case points estimation. In this paper it is presented a strategy to generate test scenarios for unit testing and integration testing, based on use case model. Such test scenarios are generated automatically through a module that uses information from this broader environment and they are a starting point to test case definition. The data modeling was based on some proposals found in literature which provided the means to implement the dependency chart, graphs and the test tree. Such structures supported the test scenarios generation. Based on a system previously tested from a real company, we compared the unit and integration test scenarios generated by the environment and the results showed that some test cases were not covered by our new module but, on the other side, some generated scenarios were not planned by the company. With this in mind, the test scenarios automatically generated by the environment can be considered as a support to the test plan elaboration and to the test case definition.

Keywords: Test scenarios generation; unit test scenarios; integration test scenarios; automated environment; use case

1 INTRODUCTION

Ensuring the quality of software products is something increasingly important in the context of computer systems development. Thus, it is essential that a set of supporting activities, called Software Quality Assurance, are applied during the process of software development. These activities consist especially in testing and inspection activities, which aim at evaluating the various software artifacts that are developed throughout the process such as the requirements document (RD), the project models, the test plan, and so on.

Requirements engineering is one of the steps of the development process in which various defects can be unintentionally inserted. Thus, the understanding of the user requirements becomes a critical factor in the success of a software project, as well as in the correct interpretation of these requirements in the RD and, afterwards, in some other models, as in the Use Case Model (UCM).

Therefore, techniques should be applied to support the development and the validation of the artifacts generated by this step. To achieve this goal, this research group has developed the TUCCA technique - Technique for Construction and Use Case Requirements Document Analysis [1] – and the COCAR environment, which provides automated support for this technique [2]. This environment, which focuses on the use case, is also used to support other development activities found in the software development lifecycle.

In literature we can find some initiatives for applying VV&T activities based on use cases. Most of them aim at generating test scenarios based on the use case model and some of these have supported this paper.

This paper presents how the unit and integration test scenarios based on use case model were generated and how they are available in COCAR environment. The paper is organized as follows: Section 2 presents the COCAR environment and Section 3 presents related works in the context of VV&T and Use Cases. Section 4 illustrates how the test scenarios have been built and how they are available in COCAR environment and Section 5 presents a case study. Section 6 concludes the paper.

2 THE COCAR ENVIRONMENT

The COCAR environment has been developed to cover various steps of the software development lifecycle by supporting some activities and focusing on the UCM. According to Munson and Nguyen [3], the greater the number of steps involved in the software development process a unique tool covers, the better the results it provides, since it can avoid integration mistakes, favor the traceability of the artifacts generated and increase the quality of the process. Based on this, the previous version of the COCAR environment, without the VV&T activities proposed in this paper, consisted of:

1. System requirements insertion based on a defined template [1], which allows TUCCA to be applied in an later step;
2. The construction of the use case model: based on the previously inserted requirements, the environment supports the use of TUCCA [1]. This technique contributes to the construction of higher quality standardized UCMs by following well defined guidelines and providing an RD inspection [1];
3. Automatic Use Case Points estimation;
4. Support to Requirements Management focusing on the traceability between the requirements and the use cases in which they are reflected.

3 RELATED WORK

In literature we have found some initiatives that use the UCM as a starting point to generate information for testing [4][5][6][7][8][9][10][11][12]. Some of the related works present the generation of unit test scenarios while others show the generation of integration test scenarios.

Ryser and Glinz proposed the SCENT method [6] [7] for helping developers to create test scenarios by using artifacts created in the beginning of the software development process. Such artifacts consist of the use case diagram and the specification of each use case depicted in the diagram in natural language.

Briand and Labiche [8] proposed an approach for supporting the derivation of system test cases. First, a modified activity diagram is created (UML style) which establishes a temporal relationship between the use cases and the system classes. After that, this activity diagram is transformed into a graph which is, then, used to build a test tree with each branch representing a path in the graph. Finally, a test case should be elaborated for each branch of the tree considering the parameters specified in the use cases.

Carniello [9] defined structural test criteria that establish test requirements from the use case diagram. The use of these criteria provides coverage information, which allows the test activity to be quantifiable. The structural elements considered for defining the criteria are: communication relationships between actors and use cases, inclusion relationships (<<include>>) and extension relationships (<<extend>>). To support the application of the proposed set of criteria, a test coverage tool, called Use Case Tester – UCT, has been developed. Linked lists are used to store and organize such execution sequence.

The proposal of Hartmann et al. [4] to automatically generate and execute system test cases is UML-based and considers behavioral models of an application The authors developed a graphical user interface (GUI) tool in order to execute the generated test cases. Firstly, UML models, mainly activity diagrams, are semi automatically extracted from the use case documentation. Then, a test designer manually annotates the UML models with its test requirements. After that, test cases or test scripts are automatically generated by a tool. Finally, a test executor runs the test cases or scripts using some commercial UI testing tool. The authors introduced the concept of generating

Nebut et al. [10] proposed a methodology to generate functional test scenarios automatically using two distinct phases. The first phase consists of generating the test objectives from the UCM. The second phase consists of generating test scenarios from the test objectives determined in the previous phase. The transformation process from test objectives to test scenarios is done through message exchanges between the environment and the system. Such message exchanges should be represented using UML diagrams, like the activity diagram, the sequence diagram or state diagram.

These works are related to Model-Based Testing (MBT) paradigm. MBT is a general term that signifies an approach that bases common testing tasks, such as test case generation and test result evaluation, on a model of the application under test [17].

We proposed a combination of the approaches detailed in this section in order to generate unit test scenarios and integration test scenarios in COCAR environment. Besides, the solution proposed has been integrated to the other functionalities existing in COCAR, such as the maintenance and traceability of requirements, building of

UCM following the TUCCA guidelines, and the automated use case points estimation.

4 UNIT AND INTEGRATION TEST SCENARIOS GENERATION

In this section we present how the unit test scenarios and the integration test scenarios were generated based on a combination of the related works presented in Section 3. In the context of this research, a unit test scenario refers to a specific use case, while an integration test scenario refers to relationships between the use cases.

The original COCAR environment required the use of the TUCCA technique to specify the use cases. In order to allow users to insert the use cases in COCAR without applying TUCCA, two other functionalities have been added to COCAR: (i) manual entry of the use case specifications, which consist of the normal flow, the alternative flow and the dependency relationship between the use cases (pre and post conditions, include and extend relationships); (ii) use case import from XMI (XML Metadata Interchange) files [13] generated by another modeling tool, as ArgoUML [14]. In this case, essential information for the test scenarios generation (like the dependency relationship between the use cases) must be provided by the user.

Figure 1 illustrates the combination of the related works described in Section 3 which resulted in a new module in COCAR responsible to generate the test scenarios. The additional modules developed to aid manual registration of use cases and XMI import are also represented.

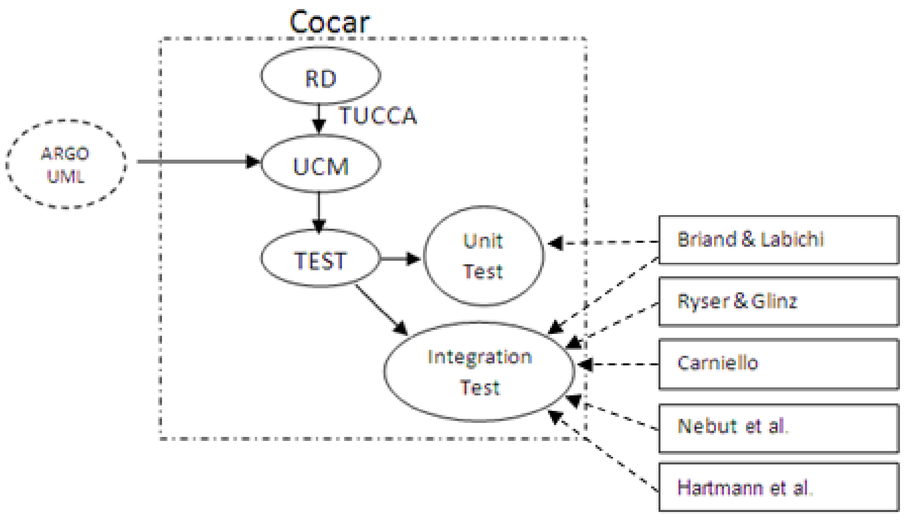


Fig. 1. An overview of how related works were used in the COCAR testing strategy.

4.1 Unit Test Scenarios Generation

The unit test scenarios generation is based on the work proposed by Briand and Labiche [8], from which it was used the concept of building a graph and, from it, deriving a test tree. In the context of this research, a graph represents the distinct paths existing for a use case when considering the normal and the alternative flows. The test tree, which is derived from the graph, represents each possible path individually.

The first step to build the graph for a chosen use case is to add a node for each step of the normal flow. Thus a sequence of linked nodes would be created, representing one of the possible scenarios for the use case. Once this step is done, the next step is to add the nodes for each alternative flow related to the use case. These nodes should be added as alternative paths for the specific nodes representing the normal flow that they refer to in the graph. After all the alternative flows have been added to the graph, we should have a complete graph, representing all the possible scenarios regarding to the use case.

Besides, the generation of the unit test scenarios follows the concepts of Basis Path Testing (Mc Cabe's criteria [15]). This criterion has been used in order to be sure every path would be tested at least one time, and to make the implementation easier, avoiding a very complex combination of multiple paths. Hence, each alternative flow is handled by a distinct unit test scenario.

Figure 2 shows the graph related to a use case that contains a normal flow with 8 steps, and two alternative flows referring to steps 2 and 6 of the normal flow, respectively. In this case there are three unit test scenarios: (i) the path 1-2-3-4-5-6-7-8; (ii) 1-2.1-2.2-2.3; and (iii) 1-2-3-4-5-6.1-6.2-6.3-7-8.

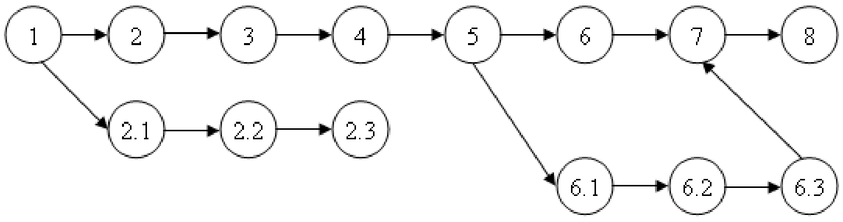


Fig. 2. Graph containing normal and alternative flows of a specific use case.

4.2 Integration Test Scenarios Generation

The integration test scenarios generation are modeled based on the SCENT method [6] [7] mainly. Based on the use case diagram and provided their execution order is known, the method suggests the creation of a dependency chart that shows the dependencies between the use cases. Through the use of a specific notation, the dependency chart models elements like the sequence, the iteration, the concurrency and the dependencies related to time.

The logical dependencies modeled in the dependency chart are determined based on the use case specifications, considering the inclusion (<<include>>) and extension (<<extend>>) relationships, and also on the logical dependencies among the use cases that can be found in the pre and post conditions of the a case specification.. After modeling all the logical dependencies, a graph is derived from this dependency chart. The initial nodes of the graph should be the use cases (one or more than one) that have no logical dependencies to be executed. From them, based on the logical relationships modeled in the chart, it is possible to determine all the paths of the graph.

From this graph is derived a test tree containing the valid scenarios of a system, as suggested by Briand and Labiche [8]. Each branch of the test tree should be an integration test scenario that is supposed to be tested. A similar approach can be found at the proposal by Nebut et al. [10].

Figure 3 shows an example of graph built based on the dependency chart, relating use cases.

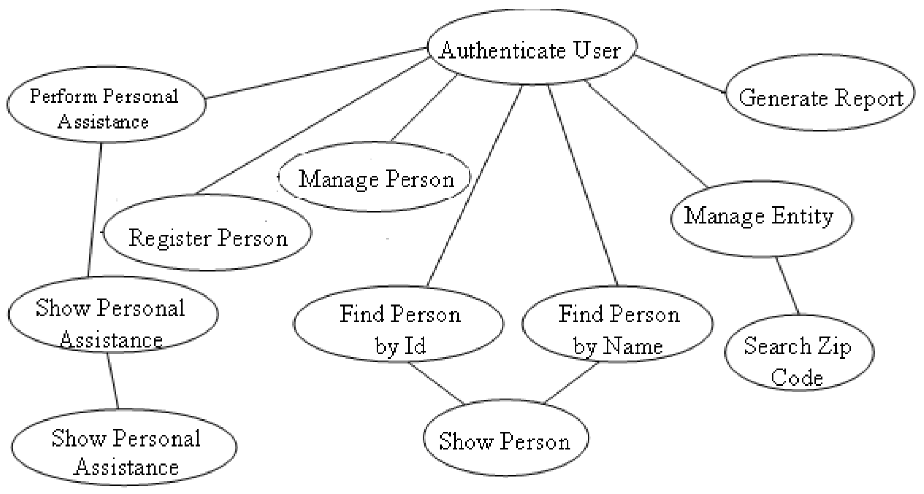


Fig. 3. Graph built based on the use case dependencies.

From Carniello [9] it was taken the definition of a structure for storing use case information with a difference that relational tables were used instead of linked lists. This approach has been used to store and access the necessary information in order to generate the test scenarios.

From Hartmann et al. [4] it was used the idea on how to integrate the environment COCAR with other modeling tools. A difference is that we used XMI files to integrate COCAR to other tools, while the authors used XML as input for a tool developed by them. This integration is important to allow a use case modeled in another tool could be imported and used into COCAR.

What distinguishes this work from the related ones is that it is a combination of other works and also allows the generation of both scenarios (unit and integration) in the same environment. It is possible to work with a single use case, providing test scenarios for it (unit test scenarios), and also to work with many use cases, providing test scenarios to check the relationships existing among them (integration test scenarios).

Besides, it is possible to apply other functionalities to the use cases inserted or imported into COCAR, like automatic use case points estimation, and support to requirements management. It is possible because the new unit and integration test scenarios generation module was implemented and integrated to the other existing funcionalities.

5 CASE STUDY

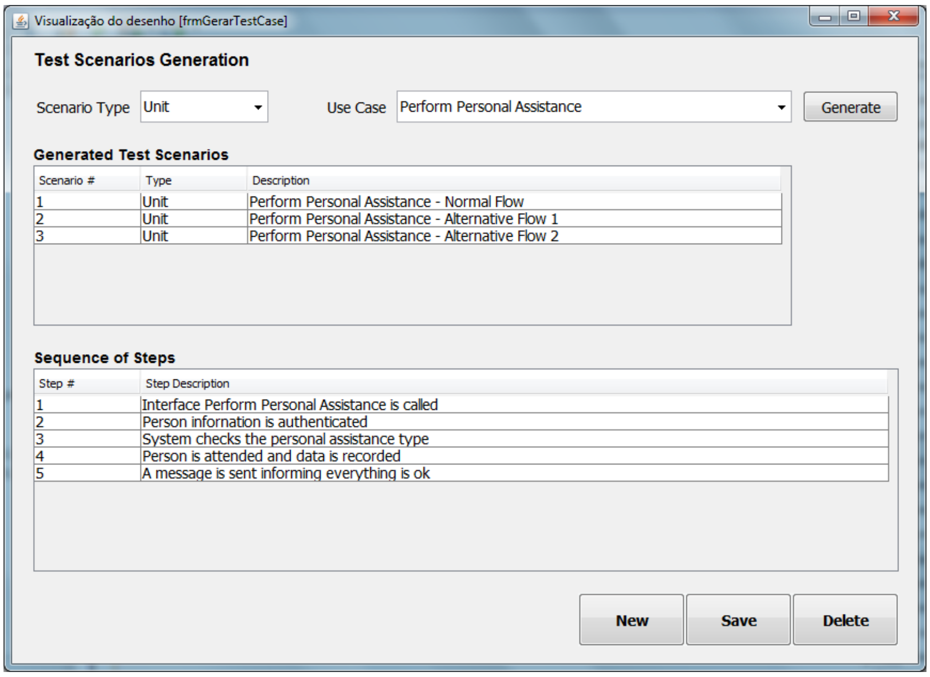
To evaluate this work, a case study has been carried out. Real company data could be taken from the site http://www. softwarepublico.gov.br [16] with previous authorization. The documentation accessed was composed by the use case diagram, the use case specifications, a table with the actors and the test plan which, in turn, contained the test cases for each existing use case. The method used for the definition of the testing plan was not informed by the company.

It is important to mention that, as the use cases already existed and belonged to a system in a production environment, they were inserted in COCAR from an XMI file or manually. The documentation had 48 use cases and 12 of them (25%) had been selected to the case study.

The procedures adopted are described as follows:

1. Insertion of 11 use cases through the use case manual registration.
2. Insertion of 1 use case modeled with ArgoUML and imported to COCAR through an XMI file.
3. Unit test scenarios generation in COCAR environment.
4. Analysis of the results obtained by comparing the unit test scenarios generated in COCAR environment with the test cases of the company’s test plan.
5. Integration test scenarios generation in COCAR environment.
6. Analysis of the results obtained by comparing the integration test scenarios generated in COCAR environment with the original test cases of the company’s test plan that corresponded to the scenarios generated.

A screen shot from unit scenario test generation in COCAR is shown in Figure 4. The use case “Perform Personal Assistance” was chosen and its unit test scenarios were generated based on its normal and alternative flows. The list named “Generated Test Scenarios” shows three possible unit test scenarios. The first is related to the normal flow, and the other two as related to alternative flows. Once a unit test scenario is selected, its sequence of steps is shown in the list named “Sequence of Steps”.


Visualização do desenho [frmGerarTestCase]
Test Scenarios Generation
Generate
New
Save
Delete

Scenario Type: Unit

Use Case: Perform Personal Assistance

Generated Test Scenarios

Scenario #	Type	Description
1	Unit	Perform Personal Assistance - Normal Flow
2	Unit	Perform Personal Assistance - Alternative Flow 1
3	Unit	Perform Personal Assistance - Alternative Flow 2

Sequence of Steps

Step #	Step Description
1	Interface Perform Personal Assistance is called
2	Person information is authenticated
3	System checks the personal assistance type
4	Person is attended and data is recorded
5	A message is sent informing everything is ok

Fig. 4. Unit Test Scenario Generation

For each use case, the data analysis carried out in Step 4 was registered in a table where columns 1 and 2 identified the scenarios generated by COCAR. Columns 3 and 4 identified the test cases of the company's test plan, associating them to the test scenarios generated by COCAR, when appropriated. Column 5 registered possible divergences, such as test cases applied by the company that were not possible to be generated by COCAR environment since there was missing information in the UCM provided by the company. An example of this analysis is shown in Table 1.

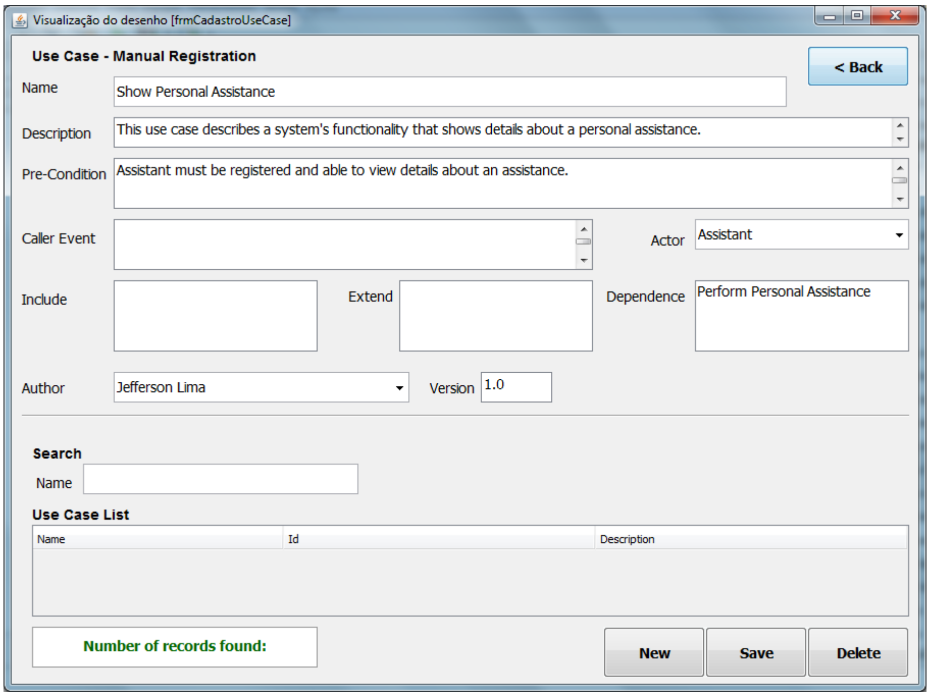
Table 1. Comparison between the unit test scenarios and the test cases related to a use case

Use Case: Perform Personal Assistance				
Scenario	**Scenario Description**	**Test Case**	**Test Case Description**	**Divergencies**
1	Normal Flow: Perform Personal Assistance	39.1 / 39.2 / 39.3	Perform Personal Assis-tance succesfully	-
2	Identify an invalid person	39.6	Identification of an invalid person	-
3	Use invalid Personal Assistance information	39.5	Invalid Personal Assisten-ce option	-
-	-	39.4	User without access per-mission	Not predicted in the use case flows
-	-	39.9 to 39.20	Security and confidentia-lity tests	Not predicted in the use case flows

For all the other 11 use cases, the same kind of comparative analysis had been carried out and the results can be summarized as follows:

- There were test scenarios generated by COCAR that referred to all the test cases of the company elaborated for a use case. This happened for 2 out of 12 use cases considered.
- There were test scenarios generated by COCAR that did not present corresponding test cases in the company's test plan. Such test cases could potentially identify software errors in advance.
- There were test scenarios of one use case provided by the company that did not present test cases to cover all possible execution paths. COCAR, on the other hand, generated test scenarios that would help constructing test cases that exercise all execution paths.

Figure 5 illustrates the screen where all the dependences among use cases are informed. There are three lists called "Include", "Extend", and "Dependence". The first one is used to inform all the use cases that are related to the selected use case ("Show Personal Assistance") by inclusion relationship. The second one is used to inform all the use cases that are related to the selected use case by extension relationship. Finally, in the last one, it is necessary to inform what use cases must be executed before the selected one, meaning the logical dependence between them. All of these dependences are used to build the dependency chart.


Visualização do desenho [frmCadastroUseCase]

Use Case - Manual Registration

Name	Show Personal Assistance
Description	This use case describes a system's functionality that shows details about a personal assistance.
Pre-Condition	Assistant must be registered and able to view details about an assistance.
Caller Event	
Actor	Assistant
Include	
Extend	
Dependence	Perform Personal Assistance
Author	Jefferson Lima
Version	1.0

Search

Name

Use Case List

Name	Id	Description

Number of records found:

Fig. 5. Graph built based on the use case dependencies.

There were some company’s test cases for which no test scenario was generated by COCAR due to the lack of specifications.

The data analysis carried out in Step 6 was registered in a table where column 1 corresponds to the integration test scenarios generated by COCAR and column 2 shows the comparison of the scenario with the company’s test cases. In this comparison, both the preconditions of the use cases and the inclusion (include) and extension (extend) relationships have been taken into account. An example of this data is shown in Table 2.

Table 2. Comparison of the integration test scenarios with the test cases of the company test plan

Scenario	Evaluation of the Company’s Test Plan
1	The relationship between the use cases "Authenticate User" and "Perform Personal Assistance" is referred in the test case 9.4 and also in the preconditions of the use case "Perform Personal Assistance". However, there is no test case to "Visualize Personal Assistance" and "Visualize Personal Assistance Details".
2	The relationship between use cases is found in the test case preconditions of the use case "Register Person", when the need for user authentication is emphasized, which satisfies the test scenario completely.

Figure 6 shows the seven integration test scenarios derived from the graph in Figure 3. They represent the valid integration test scenarios generated by COCAR environment.

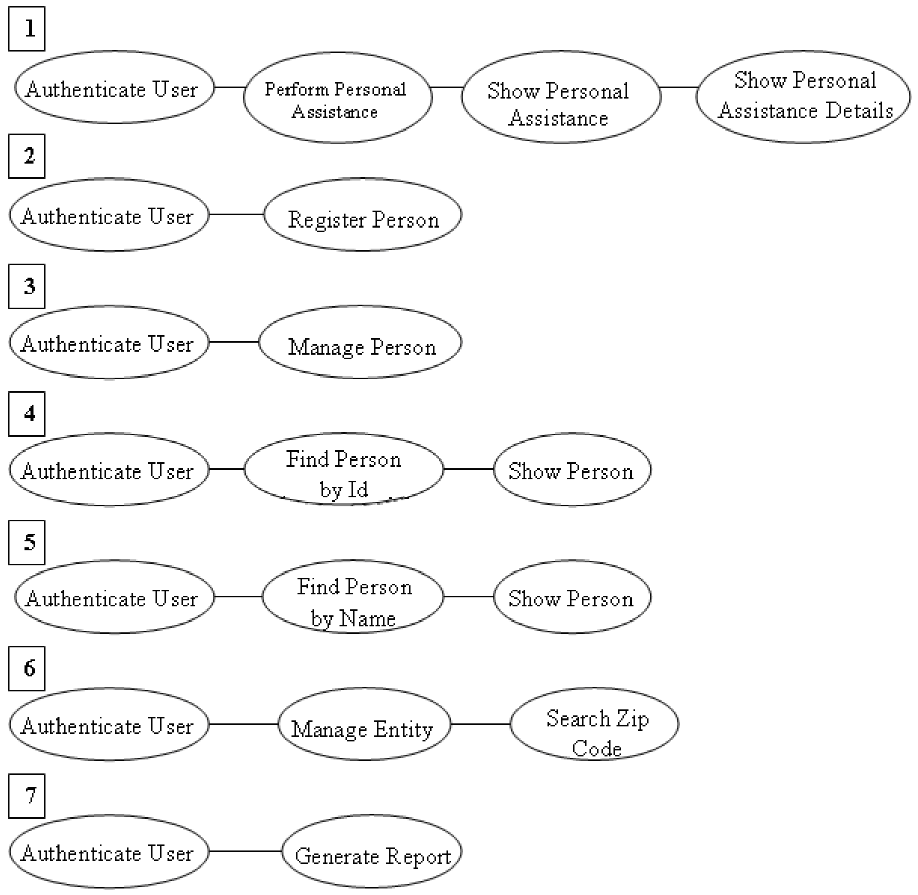


Fig. 6. Integration test scenarios derived from the graph in Figure 3.

Some conclusions have been taken from the analysis of all the company's test documentation:

- 5 out of 7 integration test scenarios generated by COCAR for the 12 use cases of the selected sample were fully covered by the company's set of test cases.
- The 2 scenarios that were not fully covered corresponded to paths (dependencies between the use cases) that were not identified by the company.

6 CONCLUSIONS AND FUTURE WORK

This paper presented a proposal for the unit and integration test scenarios generation based on UCM. The solution adopted has been inspired by other related works that use the UCM as a starting point for creating test scenarios and test cases.

As test activity (including the test plan elaboration) is usually complex and requires a great effort from the test planners, we decided to automate the process of generating test scenarios. As it already had other functionalities based on UCM, the COCAR environment has been used to have this new module added to generate unit and integration test scenarios automatically,.

A case study with real company data has been carried out with test scenarios being generated automatically by COCAR, and used to evaluate the quality of the company's test cases created through an ad-hoc approach. The results evidenced this research could contribute positively to the test plan definition, as some test scenarios generated by COCAR were not covered by the test cases found in the company's test plan.

When considering large and complex systems, with a large number of use cases and scenarios, the test scenarios generated automatically by COCAR should be a good contribution to test planners in order to create more complete test plans. Such test plans would contain test cases which cover each scenario at least once.

Through the addition of this new module to COCAR, the environment has been improved, covering another important phase of the software lifecycle – the test activity. Another advantage is that it has been designed to work integrated to the other modules existing prior to this research. COCAR has also become more flexible, since integration to another modeling tool is now allowed. This can be considered as an additional contribution of the research presented in this paper.

As future work, new case studies will be conducted to evaluate the use of the COCAR environment in industry, involving other companies and different systems. Besides, we plan to: (i) improve the integration test scenario generation module in order to contemplate also invalid integration test scenarios; (ii) generate test cases based on the use cases specification, such that they cover the scenarios generated by the COCAR environment; and (iii) evaluate the coverage of the test cases once they are generated in COCAR.

7 REFERENCES

1. Belgamo A., Fabbri S. C. P. F., and Maldonado J. C. TUCCA: Improving the Efectiveness of Use Case Construction and Requirement Analysis". In International Symposium on Empirical Software Engineering (ISESE), Australia, 2005.
2. Di Thommazo A., Martins M. D. C., and Fabbri S. C. P. F. Requirements Maagement in COCAR environment (in portuguese). In Workshop de Engenharia de Requisitos (WER 07), Canada, 2007.
3. Munson E. V., and Nguyen T. N. Concordance, conformance, versions, and traceability. In Proceedings of the 3rd international workshop on traceability in emerging forms of software engineering, USA, 2005.

4. Hartmann J., Vieira M., Foster H., and Ruder A. A UML-based approach to system testing. In Innovations Syst Softw Eng, USA, 2005.
5. Vieira M., Leduc J., Hasling B., Subramanyan R., and Kazmeier J. Automation of GUI Testing Using a Model-driven Approach. In First International Workshop on Automation of Software Test (AST'06), 2006.
6. Ryser J., and Glinz M. SCENT: A method employing scenarios to systematically derive test cases for system test. Thesis, University of Zurich, Switzerland, 1999.
7. Ryser J., and Glinz M. Using Dependency Charts to Improve Scenario-Based Testing. In 17th International Conference on Testing Computer Software (TCS), USA, 2000.
8. Briand L., and Labiche Y. A UML-based Approach to system testing. Technical Report, Carleton University, Canada, 2002.
9. Carniello A. Test based on the use case structure (in portuguese). Thesis, Unicamp, Brazil, 2003.
10. Nebut C., Fleurey F., Traon Y. L., and J. M. Jézéquel. Automatic Test Generation: A Use Case Driven Approach. In IEEE Transactions on Software Engineering. Vol. 32, Nº 3, 2006.
11. Fröhlich P., and Link J. Automated Test Case Generation From Dynamic Models. In Proceedings of the 14th European Conference on Object-Oriented Programming (ECOOP '00), 2000.
12. Katara M., and Kervinen A. Making Model-Based Testing More Agile: A Use Case Driven Approach. In Proceedings of the Haifa Verification Conference, 2006.
13. Unified Modeling Language Superstructure. Available at http://www.omg.org/technology/documents/formal/uml.htm. Last access on November, 2010
14. ArgoUML documentation. Available at http://www.argouml.tigris.org. Last access on December, 2010.
15. Pressman R. S. Software Engineering, 6th ed. McGraw Hill Companies, USA, 2006.
16. Public Software documentation. Available at http://www.softwarepublico.gov.br. Last access on December, 2010.
17. Ibrahim K. El-Far and James A. Whittaker. Model-Based Software Testing. In: Encyclopedia on Software Engineering, Wiley, 2001.