An Industrial Experience in Comparing Manual vs. Automatic Test Cases Generation

Francesca Basanieri
ISTI – CNR Pisa
Via Moruzzi 1
56124 Pisa
francesca.basanieri@isti.cnr.it

Pierpaolo Iani
Ericsson Lab Italy SpA
Via Cadorna, 73
20090 Vimodrone Milano
pierpaolo.iani@eri.ericsson.se

Gaetano Lombardi
Ericsson Lab Italy SpA
Via Anagnina, 203
00040 Roma
gaetano.lombardi@eri.ericsson.se

Eda Marchetti
ISTI – CNR Pisa
Via Moruzzi 1
56124 Pisa
eda.marchetti@isti.cnr.it

Abstract.

We present our experience in automatically deriving a detailed test case plan exclusively using the UML diagrams developed during the analysis and design phases. We consider in particular the Integration Testing of some new functionalities to be added to an Ericsson Lab Italy (ERI) existing system. In particular we compare the obtained test plan with the "official" one, which was manually derived in an independent manner by the ERI personnel, highlighting their respective characteristics and weaknesses.

1. Introduction

In recent years, the object oriented (OO) paradigm has gained widespread use in both industry and universities . The reason for such popularity is mainly the natural correspondence between system components and objects defined by the distinctive features of OO: encapsulation and inheritance. Moreover, the graphical notation adopted by OO models facilitates system analysis and the representation of various design aspects at different levels of abstraction. UML, the Unified Modelling Language [15], is *de facto* the OO standard notation and nowadays is being widely adopted in industrial design practice for the modelling and specification of systems throughout all phases of the development process. Although a large body of literature exists on using UML in design, only a relatively small portion is devoted to its application in testing or provides specific assistance for planning and generating tests starting from UML descriptions. This paper deals with UML-based testing. Testing is clearly an important part of the software development process, which can impact heavily on the cost and reliability of the final product. Hence, it is easy to understand that the search for practical UML-based methods for improving the effectiveness of software testing has attracted ever-increasing interest in research

The guiding principle of our research is to take the same UML diagrams developed for analysis and design, and apply them, *as is*, to testing, without the need for any additional formalism or *ad hoc* mechanisms specifically for testing purposes. We have therefore proposed an original method, called Cow_Suite [2] and implemented it in a tool, for deriving meaningful test-case suites, right from the highest-level testing stages, by starting with the UML diagrams of the system in question. This paper focuses on the Integration Testing, which is a systematic process, applied to reveal problems in components interfaces and their interactions when combined, , after having been tested in isolation. Thus we illustrate the use of the Cow_Suite tool, in particular the UIT (Use Interaction Test) method [2], for automatically deriving test cases from UML diagrams[1]. For this we use a real-world case study provided by Ericsson Lab Italy SpA, ERI, involving the Integration Test of some new functional additions to an existing system. Using Cow_Suite we derived a detailed test case plan, called *UIT test plan* for the Integration Testing of these new

[1] Cow_Suite tool also implements a strategy for test cases selection and prioritisation on the base of their importance and effectiveness in the testing phase but the details of Cow_Suite are beyond the scope of this paper.

functionalities. The UIT test plan was automatically derived outside the production processes, exclusively using the UML diagrams developed during the analysis and design phases. The ERI personnel had independently derived another test plan (the “official” one), called *ERI test plan*, for the same functionalities. The ERI test plan was developed manually, following the standard in-house procedures at ERI and was based mainly on the personal expertise of the people involved and their knowledge of the system. We provide here a comparison of the two tests plans, highlighting their characteristics and respective weaknesses, focusing on different aspects of the plans, such as, for example, the time and effort necessary to draw up the two test plans or the structuring and the degree of detail of test cases attained for the same functionality in the two approaches. It is necessary to specify that it is out of the scope of this paper comparing the final test cases derived using the two approaches in terms of number of produced tests or time required to execute all them nor in terms of detected failures. We present only our experience in applying automatically the UIT method in a large-scale system, to derive test cases and construct a significant test plan; our aim is to report and to evaluate the main advantages in terms of costs, schedule and test strategy selection.

The paper is structured as follows. Section 2 presents the analysed case study while the ERI test strategy is described in Section 3. Section 4 and 5 deal respectively with the UIT method and the details of the two test plans devised. The comparison of the two different test plans follows in Section 6, while Section 7 draws some conclusions and provides some indications for future work.

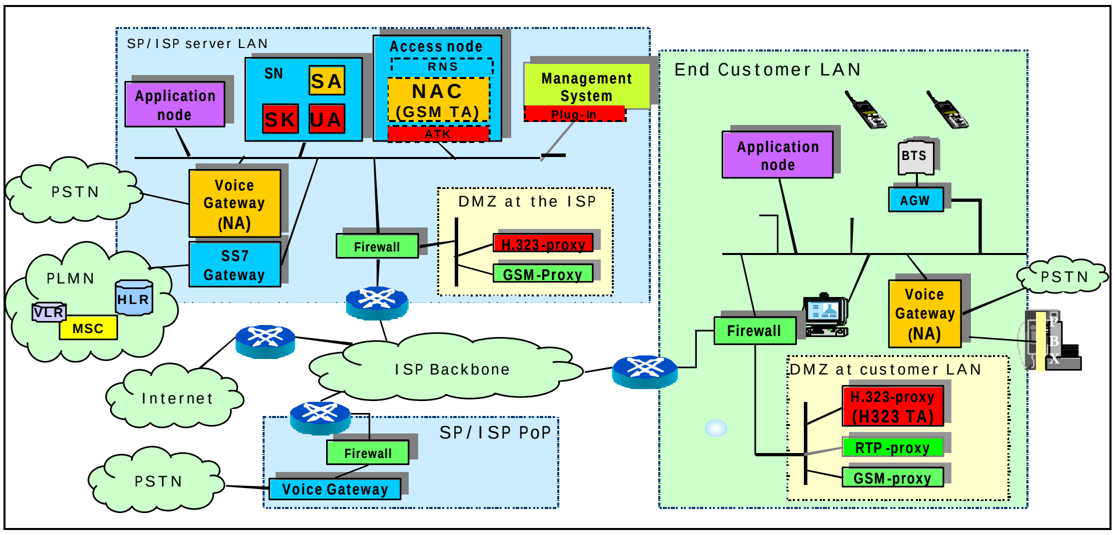

Figure 1 System Description

2. The Case Study

The case study concerned a project whose aim was to develop an IP Telephony system to support GSM communications based on the H.323 architecture [7]. Figure 1 depicts the entire system, together with the subsystems under ERI responsibility SK, UA, H.323 proxy and the associated plug-in.

The main component of the system is **Service Node** (SN) composed by: **User Agent** (UA) which implements all the users system functionalities; **UserService Agent** (SA) implementing the supplementary services, like Virtual Private Network, and Call Forwarding; **SiteKeeper** (SK), that represents the interface between the SN and the system access provider. All terminals enter in the SN through the SK, which performs the routing for the calls and resources management.

The specific feature used to compare ERI's manual vs. UIT-based automated test-case definition was the **Basic Routing Enhancements** (BRE). This feature represents an improving of the routing functionality in the GSM on the Net system, which is a new multimedia system based on the IP protocol. The upgrade regards mainly extension of some tables through the addition of new parameters and the implementation of new functionalities, for determining the enterprise or the User Agent Group associated, giving a certain number. As proper implementation of BRE implies modification of the Site Keeper and User Agent, an accurate and specific Test Plan was needed.

3. The ERI Test Strategy

To better illustrate the test strategy adopted by ERI, a brief description of the project's scope is in order. The project had to implement eleven features. After careful analysis the ERI personnel discovered that they were nearly all independent even if all components were affected by more than one feature. For this reason, the project involved the identification of a specific test strategy per feature with the aim of covering the requirements as well as the architecture of the system as a whole. The test strategy defined by the project comprises nine different testing activities (Table 1), each described in a specific Test Plan (TP). Nevertheless, not all the activities were mandatory; each feature had its own test strategy defining the test activities to be performed. The purpose of having a specific test strategy for each feature was to arrive at the best trade-off between quality and time.

Table 1 Testing Activities Description

Testing Activities	Characteristics	Responsibility
Class Test	Executed both in static and dynamic mode	Design team
Component Test (WB)	White Box (WB), aiming at testing the interfaces among classes	Design Team
Component Test (BB)	Black Box (BB), aiming at testing in a simulated environment the functions implemented by the component and its behaviour.	Design team
Node Test WB	WB, aiming at testing the interfaces among components.	Design team
Node Test BB	BB, aiming at testing in a simulated environment the functions implemented by the node and its behaviour.	Design Team
Feature Test Pre-Integration	Functional Test in a simulated environment using the real code. Its TP is derived from the detailed requirements. The main purpose is to deliver to the Integration & Validation team a feature running and clean.	Design team
Feature Test	Functional executed in the target environment. The TP is derived both from detailed requirements and main requirements.	Integration & Validation team (I&V)
Regression Test	Mandatory of the end of each feature delivery	I&V team
Performance Test, Stability Test, Negative Test, Overload Test, Characteristic Test, Capacity Test	Described in a specific Test Plan.	I&V team

In particular the BRE test strategy was to perform five different testing activities (the last common for all the features): Class Test; Feature Test Pre Integration; Feature Test; Regression Test (twice); Performance Test. For the purposes of illustration, herein we concentrate on the Pre-integration Test and present and compare the two different test plans with regard to this aspect of the BRE testing.

4. Use Interaction Test Strategy

In this section we briefly describe the Use Interaction Test (UIT) methodology [2] used to derive a set of Test Cases which is largely inspired to the Category Partition [11]. UIT systematically constructs and defines the tests for Integration Testing phase by using the UML diagrams as its exclusive reference model. In particular UIT is based on the Sequence Diagrams, which describe how a Use Case is realized by the interactions among objects and actors through elaborations and message exchanges [15]. Very briefly considering a Sequence Diagram a *Messages_Sequence* is defined considering each message with no predecessor association (see [15] for more details) plus, if existing, all the messages belonging to its nested activation bounded from the focus of control region. Thus a Messages_Sequence represents the performance of an action to be tested and describes the interaction among objects, characterizing the level of detail of integration, necessary to realize the corresponding functionalities. For every derived Messages_Sequence, UIT identifies then the *Interactions Categories*, which are the messages involved, and *Setting Categories*, which are all the possible parameters or data structures that can affect these messages. A *Test Case* is formed by a Messages_Sequence plus all the Categories involved and it is specifically designed to test the correctness of the messages interactions. Afterwards, by instantiating the categories values involved in a Test Case, a set of executable Test Procedures [14] is obtained. Thus a *Test Procedure* is defined as a set of detailed instructions for setting-up, executing, and evaluating the results of a given test case. In detail, for each Test Case the possible values that the categories can assume, called *choices*, representing: for the Interactions Categories the list of specific situations in which the message can occur; for the Settings Categories the set or range of input data that parameters or data structures can assume. A Test Procedure is directly generated for each possible combination of choices, of every categories involved in a Messages Sequence.

5. Test Plan Description

In the following sections we briefly describe the structure of the ERI and the UIT Pre Integration Test Plan.

5.1 ERI Test Plan

The ERI_TP, defined specifically for testing the BRE functionalities, is essentially a natural language document describing the test cases configuration, and the test results expected for the coverage of the BRE requirements. In drawing up the ERI_TP, the ERI designers base their decisions solely on their personal knowledge of the system, both for definition of the test cases and validation of their accuracy with the respect to the requirements. The test specifications were in fact defined "manually" according to the standard in-house procedures at ERI, without reference to the UML system description; moreover, no tool or automatic device was applied for deriving the test cases. Once the test cases were defined, each test was then assigned to a specific Test Group representing high-level system functionality and exploiting by the Project Manager to check the requirement compliance. In greater details, each test case is divided into three separate parts: *Description, Precondition* and *Procedure.*

The *Description,* defining the goal, provides a description of the environment, the entities involved in the test and the specific conditions under which the test should be run. The *Precondition* delineates, listing them explicitly in natural language, the values of data structures, the behaviour that the test case must exhibit, the actions it is to perform and the conditions required for test execution. The *Procedure* part is, in turn, divided into three sections: *Action, Result* and *Comment.* The Action consists of a brief description of the steps necessary for constructing the test case and assigning values to its variables. The Result section describes the expected outcomes. Finally, the Comment section may contain some notes or suggestions for a proper execution. In Figure 2 one of the ERI_TP developed test case is reported.

TEST GROUP 1: CALL ESTABLISHMENT WITH ENTERPRISE INFORMATION
This test group aims to verify that the Control Node is able to establish different kind of calls using the Enterprise information.
TC 1: Basic call from internal user to External Network(EN), Enterprise with PNP, Enterprise determination based on e164 alias
Description
This test is made to verify that the typical call case from user to EN works properly using the information of the Enterprise the user belongs to. The needed Enterprise determination is performed using the e164 alias in the incoming SETUP message.
Precondition
- The file *MasterRoutes.def* must contain a row looking like this: 1 (=UA) UAGname 1 (=e164 Route Type) EnterpriseName Digits 0. An example could be: 1 UAGxxx 1 Ericsson 39067258 0
- A GW is to be added to the Network Topology; this Access Agent must be associated (via a proper Access Group) to a suitable route (let's make it for example "39068") and to the Enterprise the user belongs to. The file *MasterRoutes.def* could contain for example a row looking like this: 1 Agxxx 1 Ericsson 39068…
- Another GW is to be added to the Network Topology; this Access Agent must be associated (via a proper Access Group) to the same above route but to a different Enterprise (let's make it for example "Nokia"). The file MasterRoutes.def could contain for example a row looking like this: 1 Agyyy 1 Nokia 39068…

Procedure
Action:
Make a call from the user belonging to the first Enterprise (in our example "Ericsson") to the above GW. The first digits of the dialled number must match to the above route (in our example a suitable Called Party Number could be "39068xxxx").
Result:
The final result is the call termination towards the GW reserved to the Enterprise the user belongs to (in our example to the GW in the AG "AGxxx", **not** "AGyyy").
Comment:
Only the GWs reserved to the Enterprise can be used for routing calls.

Figure 2 ERI_TP Test Case Description

5.2 UIT Test Plan

The UIT_TP is automatically derived applying the Cow_Suite tool to the UML description of the system. The methodology can applied right from the early phases of software development to define Test Cases that can be further refined with increasing level of the design's detail. The UIT_TP can be derived at two different levels of detail: at Test Cases or at Test Procedures level. At Test Cases level, UIT_TP specifications can be derived using exclusively UML diagrams developed during analysis and design phases, far before the testing phase. Test Cases construction does not require any specific knowledge of the system, because it is derived automatically from the information in UML diagrams without any additional formalism or ad hoc mechanism specifically for testing purpose. Each derived Test Case contains information useful to determining interactions of the units involved and how to test them. Once derived, Test Cases are grouped into *Use Case Test Suites* (UCTS). Every UCTS is associated to a specific Use Case and contains the Test Cases generated from all the Sequence Diagrams linked to the Use Case. The test set of an UCTS represents the actions necessary in order to check correct performance of the functionality described in the Use Case. Figure 3 shows two of the derived TC for example of UCTS 1 corresponding to the Test Group 1 of ERI_TP reported in Figure 2.

6. Comparison of Results

In this section we report the comparison between the two test plans that concerns both contents and development effort (Section 6.1 and 6.2). It is important to stress that only the ERI_TP test cases have been actually executed by ERI testers during the BRE pre-Integration testing. As said in the Introduction, in this paper we do not evaluate the effectiveness of the two plans in terms of fault detection nor time required for the real execution of test cases. We report only about the pros and cons of the UIT method in test generation, proving that it can be considered a valid instrument for defining test plans in industrial environment.

6.1 Comparison of the Content of Test Plans

Requirements coverage: In the *ERI_TP* the test cases were specifically constructed with the aim of covering all BRE requirements by expert personnel. In *UIT_TP* the coverage of the systems requirements is strictly linked to the Sequence Diagrams construction and information content. In this case quite a complete UML system description (and in particular, Sequence Diagrams specifications) was provided, consequently, as can be seen in Table 2, UIT_TP provides exactly the same requirement coverage as ERI_TP. In this table, the rows represent the different requirements of the BRE functionality; the columns labelled TG1...TG5 are the test cases groups in ERI_TP while those labelled UCTS1…UCTS8 are the set of Test Cases derived from the Sequence Diagrams associated with a specific Use Case (UCTS stands for Use Case Test Suite). An "X" in the cell signifies that a requirement is covered by a test case.

Expressiveness: **I)** some of the test cases described in the *ERI_TP* and relative to the system exceptions (see marked rows in Table 3) are not derivable via the UIT method due to the absence of the relative Sequence Diagrams. In the UML design, in fact, there are not the descriptions of the scenarios relative to exceptional behaviours. In such situation the automatic test derivation can't overcome the "incompleteness" of design; **II)** in the *UIT_TP* some test cases not provided in the ERI_TP are derived from two Sequence Diagrams that describe the same objects interaction from two different points of view. These Test Cases do not increase the requirements coverage of the UIT_TP, but represent a different way to test the same functionality.

Degree of detail: The *ERI_TP* test cases are clearly more thorough and detailed than those of UIT_TP. The former are provided by an expert designer who uses his/her experience with and knowledge of the system components and interactions. The test cases, therefore, specify in detail the steps necessary to execute the test cases and provide a complete description of the environment and expected results.

The Test Cases of the *UIT_TP* are automatically derived using the information in the Sequence Diagrams; they contain only specifications of the operations without any reference to the environment or necessary preconditions. The described situation can be resumed in Table 3 in which: the rows contains the ERI_TP test cases subdivided into groups, the columns the UIT UCTS. An "X" in the cell signifies equivalence of the test cases based on the two methodologies.

Use Case Test Suite 1	
Sequence Diagram "BRE-Step1: Call User to External Network: Originating Case/ Terminating Case"	
Test Case 1	**Test Case 2**
Description:	*Description:*
Precondition:	*Precondition:*
Flow of Event:	*Flow of Event:*
SETUP(A,B)	SETUP(A, B, Enterprise)
DetermineEnterprise(A)	LRQ(A, B, Enterprise)
HRA()	GRA(B, Enterprise)
Categories:	*Categories:*
Settings Categories:	*Settings Categories:*
A	A
B	B
MasterRoutes.def	Enterprise
Enterprises.def	MasterRoutes.def
Network Topology	Enterprises.def
Interactions Categories:	Network Topology
SETUP(..,..,..)	*Interactions Categories:*
	SETUP(..,..,..)
DetermineEnterprise(..)	LRQ(.., .., ...)
HRA()	GRA(.., …)
Post Condition:	*Post Condition:*
Comment:	*Comment:*

Figure 3 Use Case Test Suite Description

6.2 Comparison Relative to the Test Plans Development

Time evaluation: Considering an 8-hour working day, in *ERI_TP*, the Project Manager and the designers involved in the project have completed the plan description in 5 working days divided in three separate parts:

- Test Case definition (8 hours): the designer analyses the system components in order to identify the possible test cases and constructs a testing schema for each interaction that should be tested.
- Procedures definition (16 hours): the designer specifies all the steps and actions necessary in order to check the system's interactions, particularly the description of the environment and definition of parameters.

- Refinement and completion of documentation (16 hours): the designer and project manager review the ERI_TP and correct any errors or inaccuracies.

The main advantage of the UIT_TP-based approach is that it is not necessary to spend time on formulating Test Cases; these are in fact derived automatically from UML design descriptions using the Cow_Suite tool. By simply executing the tool with the UML diagrams as input, the first part of the ERI_TP development cycle is completed immediately. Completion of the UIT_TP, and therefore derivation of the Test Procedures, requires specification of the values of choices and constraints. We asked an ERI designer to work interactively with the Cow_Suite tool to insert the required information. Specifically for UIT_TP we have:

- Test Cases definition (0 hours): these are derived automatically from UML design descriptions using the Cow_Suite tool, by simply executing the tool with the UML diagrams as input.
- Test Procedures derivation (2 hours): the designer who works interactively with the Cow_Suite tool inserts specification of the values of choices.
- Refinement and completion of documentation (8 hours): the designer and Project Manager check the correctness of the derived Test Procedures and choose those to be actually run.

Table 2 Requirement Coverage Matrix of ERI_TP and UIT_TP

	TG1	TG2	TG3	TG4	TG5	UCTS1	UCTS2	UCTS3	UCTS4	UCTS5	UCTS6	UCTS7	UTCS8
RS_BRE1					X								
RS_BRE2	X	X				X	X				X	X	X
RS_BRE3	X	X				X					X	X	X
RS_BRE4	X					X	X						X
RS_BRE5	X					X							X
RS_BRE6	X					X							X
RS_BRE7			X					X					
RS_BRE8	X					X	X						X
RS_BRE9	X					X							X
RS_BRE10	X					X							X
RS_BRE11				X					X				
RS_BRE12					X					X			

Table 3 Comparison matrix for Test Coverage: the UCTS7 and UCTS8 Test Cases are an Alternative to the UIT derived Test Cases while Test Cases TG1-TC6 and TG5-TC2/3 are not Provided for in UIT_TP

ERI_TP - Test Group and Test Cases		UIT_TP - Use Case Test							
		UCTS1	UCTS2	UCTS3	UCTS4	UCTS5	UCTS6	UCTS7	UTCS8
TG1	TC1	X							X
	TC2	X							X
	TC3	X							X
	TC4	X							X
	TC5		X						
	TC6								
TG2	TC1						X	X	
	TC2						X		
TG3	TC3			X					
TG4	TC1				X				
	TC2				X				
	TC3				X				
TG5	TC1					X			
	TC2								
	TC3								

Summarizing, derivation of the executable Test Procedure involves only 10 hours' time with the UIT methodology, while 40 hours are needed for complete derivation of the ERI_TP. Although the UIT_TP-derived Test Procedures can be passed on directly to the tester for the execution, they still lack the specifications regarding environments and pre and post-conditions. Such data must be included and an additional day's work must be accounted for. Therefore, the UIT_TP requires in total 18 hours.

Development Stages: the *UIT_TP* can be defined as soon as one or more Sequence Diagrams have been produced, i.e., during the analysis or design stage. Any user, even with no particular system experience, can automatically derive the Test Cases by simply applying the UIT method with the help of the Cow_Suite tool. These Test Cases are not the final result of the UIT method application; they can be employed by the Project Manager for test scheduling and costs estimation in an early stage of project development. The derived test plan can be, in fact, used by the Project Manager for making decisions regarding the type of testing strategy to adopt (fulfilment of requirements, code coverage, testing the more peculiar functionalities) or selecting the test cases and making an initial estimation about the number of them to be implemented. The subsequent specification of

Test Procedures in the *UIT_TP* requires, instead, the designer specific knowledge of the system. Therefore, the Test Procedures can be derived only specifying the values for the Settings and Interactions Categories. The *ERI_TP* is described by two specific ERI staff members: an expert designer, who is responsible for test-case derivation, and a Project Manager, who acts as supervisor and ultimate decision-maker with regard to acceptance of a test plan. Thus, the ERI_TP can only be drawn up at the end of the design phase, just before the testing phase, because a final, detailed description of all system components is required. In this way, only when the testing plan is completed, the Project Manager can verify the degree of requirements coverage attained or the significance of the test cases derived and above all decide whether the test strategy adopted is suitable, or not.

Degree of detail of final Test Procedures: the *ERI_TP* requires the tester to decide which the appropriate values are for each test procedure in order to attain requirements coverage, and therefore how many tests to run. The Test Procedures in *UIT_TP* are, instead, more refined because the tester can examine all the combinations obtained by the insertion of choices values and directly choose those to execute. Moreover, considering the Project Manager point of view, the number of Test Procedures (derivable from each Test Case) represents "exhaustive" coverage of the input domain with the respect to the inserted values. He/She can therefore quantify the dimensions of the test domain for each specific system interaction before the testing phase, and evaluate the number of tests run for each Test Case with respect to "exhaustive" coverage, after test execution.

7. Related Work

With the widespread acceptance of the UML standard for describing object-oriented designs, UML-based testing has become a growing field of research. In particular, some researchers have focused their efforts on finding methods and tools for guiding the stages of testing by using UML descriptions. In this section we provide a brief overview of the literature, presenting the main solutions derived both from the academic environment. In particular we differentiate them into two groups: those which require for test cases derivation translation of the UML diagrams into an intermediate formal description and those which requires annotation of the UML diagram with further (formal) information.

Considering the former group the tool UMLAUT, [6] can be used to manipulate the UML representation of a system and automatically transform it into an intermediate form suitable for validation. Similarly in [10] the authors present a formal approach to derive test cases from UML Statecharts. Other relevant proposals include: [16] which describes an approach to black-box test-generation in which an AI (artificial intelligence) planner is used to generate test cases from UML Class Diagrams; [4] which proposes a probabilistic method, called statistical functional testing, for the generation of test cases from UML state diagrams, using transition coverage as testing criterion; [8] which describes a powerful methodology for scenario-based speci?cation of reactive systems.

Considering the latter group the relevant proposals include: JUnit [9], which provides a simple framework for software unit testing [16]; the TOTEM approach [3] which supports the derivation of functional system test requirements, which are then used to then to derive test cases, test oracles and test drivers by using the sequence or collaboration diagrams associated to each use case; SCENTOR [17] which supports the generation of scenario-based testing using JUnit as a basis; AGEDIS which is a project focused on the automation of software testing, improving software quality, and reducing of the expense of the testing phase; [12] which presents a method and a tool for automated synthesis of test cases from generic test scenarios and a design model of the application. SeDiTeC [5] which supports specification of several test case data sets for each sequence diagram and automatically generates test stub for the classes and methods whose behaviour is specified in the sequence diagrams.

UML-based integration testing is clearly a field under continuous development and refinement. In this respect, all the aforementioned studies appear interesting and must be viewed as complementary to our approach. In fact, what we develop through the procedures described herein are test cases from SDs scenarios. In particular, we do not aim at formalization: our guiding criteria are rather systematicity and timeliness. Finally, our method, like the [10] approach, can be exploited in the early stages of software development to automatically generate tests from the software design.

8. Conclusions and Future Work

In the last years the research interest of the testing community has focused more and more on the use of UML for testing purposes. Several methods have been proposed for deriving test suites using UML diagrams as a reference. However, little effort has been spent so far to investigate the application of the methods to real-world industrial case studies or to carry on empirical evaluations about their effectiveness or about the effort required for their adoption. The purpose of this paper is precisely to investigate such issues, which are in our opinion of utmost importance. In fact, UML-based testing is an attractive notion, but its usefulness has yet to be demonstrated in practice. We presented UIT, an original

UML-based method for test case derivation, and employed it to build a detailed test plan, the UIT_TP, within one of the crucial phases of the ERI system process, the Pre-Integration Test. We then presented several evaluations originating from the comparison between UIT_TP and the "official" ERI test plan, the ERI_TP, built by the ERI personnel following the standard in-house procedures. ERI has been in fact certified at CMM level 3 [13], therefore the test strategies that we compared to UIT are effective and well established.

The comparison brought us to some interesting conclusions about the efficacy of the UIT method. The main advantage of UIT application was felt to be the fact that the Program Manager can exploit the provided UIT_TP as a baseline to adopt the most appropriate test selection strategy. UIT provides, in fact, the Project Manager with a detailed test plan already during the analysis or design phase, i.e., early in advance with respect to the testing stage. Therefore, the Project Manager can get a realistic evaluation of the requirement and functional coverage that can be reached. If the values predicted are not satisfactory, corrective actions can be taken or a different choice of the proper test strategy for the testing phases can be considered. Moreover, the automated derivation of UIT_TP lets to considerably reduce the time necessary for test plan completion. In the proposed case study, we estimated a reduction of the time needed for the UIT_TP derivation of four times, while obtaining the same level of requirement coverage of the ERI_TP. On the negative side, we observed that the automatic derivation of test cases failed to include the exceptional test cases, i.e., test cases to handle abnormal system behaviour. It would be opportune, therefore, that before deployment, the UIT_TP is checked by an expert and additional test cases are possibly included to cover these special situations. This need is indeed common to any other automatic test case derivation method.

In conclusion, UIT and the associated Cow_Suite tool have been quite favourably received within the ERI Company. Here we presented the application of UIT only to the Pre-Integration testing phase of the BRE test plan, but similar experiments could be replicated for the other testing phases. In the next future, ERI intends in fact to apply the methodology in other test steps. On the IEI-CNR side, we are still continuing to refine UIT, as well as the Cow_Suite tool, to better reply to the exigencies and constraints coming from the industrial users.

References

[1] AGEDIS. Available at http://www.agedis.de/index.shtml

[2] F. Basanieri, A. Bertolino, E. Marchetti, "The Cow_Suite Approach to Planning and Deriving Test Suites in UML Projects", *Proc. Fifth International Conference on the Unified Modeling Language - the Language and its applications UML 2002*, LNCS 2460, Dresden, Germany, September 30 - October 4, 2002, p. 383-397

[3] L.C, Briand, Y. Labiche, "A UML-Based Approach to System Testing". J*ournal of Software and Systems Modeling (SoSyM)* Vol. 1 No.1 2002 pp. 10-42.

[4] P., Chevalley, P. Thévenod-Fosse, "Automated generation of statistical test cases from UML state diagrams" *25th Annual International Computer Software & Applications Conference (COMPSAC'01)*, Chicago (USA), 8-12 October 2001, pp.205-214

[5] F., Fraikin, T. Leonhardt, "SeDiTeC - Testing Based on Sequence Diagrams" *In Proceedings of the 17th IEEE International Conference on Automated Software Engineering*, Edingburgh, September 2002

[6] J.M, Jézéquel, A., F. Le Guennec, Pennanech, "Validating Distributed Software Modeled with UML" *Proc. UML98*, in LNCS 1618, 365-376.

[7] H.323 Standard http://www.microsoft.com/windows/NetMeeting/Corp/reskit/Chapter11/default.asp

[8] D., Harel, R. Marelly, "Specifying and Executing Behavioral Requirements: The Play In/Play-Out Approach", *Software and System Modeling (SoSyM)*, 2003

[9] Object Mentor, Inc., JUnit, Testing Resources for Extreme Programming, http://www.junit.org (Feb. 2001)

[10] J., Offutt, A. Abdurazik, "Using UML Collaboration Diagrams for Static Checking and Test Generat". *UML 2000*, University of York, UK, 2-6 October 2000

[11] T.J., Ostrand, M.J. Balcer, "The Category Partition Method For Specifying and Generating Functional Tests". *Communication of the ACM*, 31(6), June 1988, 676-686.

[12] S., Pickin, C., Jard, Y., Le Traon, T., Jéron, J.M., Jézéquel A., Le Guennec, " System test synthesis from UML models of distributed software" In D. Peled and M. Vardi, editors, *Formal Techniques for Networked and Distributed Systems - FORTE 2002*, LNCS, Houston, Texas, November 2002

[13] M. Paul, CMM v2.0 Draft C, 22 Oct. 1997

[14] Rational Unified Process version 2000.02.10. Rational Software Corporation. On-line at http://www.rational.com/products/rup

[15] UML Documentation version 1.5 Web Site. On-line at http://www.omg.org/technology/documents/formal/uml.htm

[16] A., Von Mayrhauser, R.; France, M.; Scheetz, E. Dahlman, "Generating test-cases from an object-oriented model with an artificial-intelligence planning system", *IEEE Transactions on Reliability*, Vol. 49, Issue 1, 2000, Pp 26-36

[17] J. Wittevrongel, F. Maurer, "Using UML to Partially Automate Generation of Scenario-Based Test Drivers". *OOIS 2001*, Springer, 2001