Requirement-based software testing with the UML: A systematic mapping study

Nesa Asoudeh
Carleton University, Dept. SCE
Ottawa, Canada
nasoudeh@sce.carleton.ca

Yvan Labiche
Carleton University, Dept. SCE
Ottawa, Canada
labiche@sce.carleton.ca

Abstract- Our goal is to determine the current state of the art in requirement based testing in a UML context. We combined an automated search in digital libraries with a manual search in related journal and conference venues. The search resulted in about 1,300 papers. After applying inclusion/exclusion criteria, we selected 100 papers as our final set of primary studies. Classification results based on several criteria lead us to interesting observations such as: A small proportion of the primary studies evaluate techniques through experiments and one-third of the techniques are simply illustrated with an example; More advanced selection criteria exist in literature than those used in primary studies.

Keywords- Requirement based testing; systematic mapping study; model driven development; requirement engineering; Unified Modeling Language.

I. Introduction

At least half of the effort to develop a working program is devoted to testing [4]. Undetected errors in software can cause substantial financial loss or even catastrophic results in safety critical systems. Detecting faults as early as possible during the software development is an effective means of reducing testing cost since the cost of fixing an error increases with the time between its introduction and detection.

Requirement-based testing (RBT) aims at starting testing-related activities as early as possible during software development, specifically deriving test cases (or test case specifications) from the requirements of the software under test [1]. RBT addresses two major issues: first, validating that the requirements are correct, complete, unambiguous and logically consistent; and second, designing a necessary and sufficient, from a black box point of view, set of test cases from those requirements to ensure that the design and code fully meet those requirements. Some of the benefits of and reasons for RBT are: (1) Creating tests early; (2) Allowing test engineers to find inconsistencies and ambiguities in requirements; (3) Leading to test data independent of any particular implementation; (4) Allowing conformance testing; (5) Reducing testing costs, since testing starts early; (6) Reducing software time to market.

These suggest that RBT can be an efficient and effective approach to software verification and validation. Note also that RBT is mandatory in some software development projects, e.g., airborne software developed according to the DO-178B/C standard [23].

There are numerous methods for representing software requirements (e.g., formal specification, plain text, the Unified Modeling Language) and more than one of them is typically used [7]. As a result, there are many approaches to RBT and, in spite of a great deal of research, to the best of our knowledge, there exists no systematic literature review [8] on this topic. There have been some surveys (e.g., [13, 14]) on model-based testing (MBT), a testing activity that aims at (automatically) deriving test cases from a model specifying the intended behaviour of a software [18]. However, RBT and model based testing are not interchangeable. Not any model used in MBT can be used for RBT. On the other hand, a model-based approach is one possible way to elicit requirements, and therefore RBT often uses models.

This paper reports on a systematic mapping study [8] on RBT, thereby identifying and classifying the available research (papers) in this area. A systematic mapping study is an important piece of work since it provides a wide overview of a research topic and establishes if research evidence exists in the area. It also provides an indication of the quantity of the evidence, prior to conducting a systematic literature review. During a systematic mapping study a classification scheme is defined and then used in an analysis phase to determine the coverage of the categories of the scheme. Petersen et al. discuss the differences between these two analyses [16].

More specifically, we are interested in RBT in the context of a UML-based software development [15]. The reason for reducing the scope of the research is threefold: (1) the UML is now the de-facto standard for analysis and design of object-oriented software [15], (2) not reducing the scope this way would lead to more research papers than what can likely be possibly managed, and (3) not reducing the scope would lead to testing techniques of widely different nature, which would complicate the review and the comparison (e.g., RBT from a Petri net model would have different capabilities than a technique based on UML sequence diagrams simply because of the more formal basis of that modeling notation).

Section II discusses the protocol we followed in our systematic mapping study. Section III presents descriptive statistics about the search of RBT techniques. Section IV discusses the comparison criteria we used to compare the identified testing techniques and the results of the comparison. Section V discusses threats to validity. Section VI concludes the paper.

We dedicate a fair amount of the paper to the search protocol and the comparison criteria. Our intent is to disclose enough details to readers' scrutiny and allow adequate conclusions to be drawn from the identified RBT techniques, to limit threats to the validity of the results, to allow replications, extensions and comparisons in future works (by us or others) [8].

II. Review Method

We followed standard procedures [8, 16] whereby the first step is a planning activity. The most important

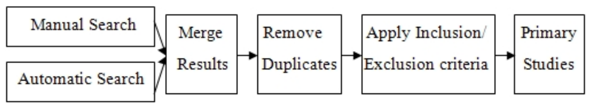


Figure 1. Paper selection process

planning activity is formulating the research questions. Our research questions are: RQ1—What are the current approaches to RBT? RQ2—What are the main characteristics of these approaches? Following planning, we identify relevant research works using an identification procedure (section II.A) and a selection/rejection procedure (section II.B).

A. Search Strategy

Most literature reviews, including mapping studies, involve automatically searching digital libraries with a set of keywords. Unfortunately, search engines embedded in digital libraries relevant to software engineering are not designed to support such reviews [5]. Therefore, one may miss relevant papers when only relying on such search engines. We complemented the automated search with a manual search, which is also recommended by others [5].

Manual Search. The focus was collecting recent papers on RBT, published between January 2006 and June 2011; We started in 2006 since UML 2.0 was officially released in July 2005 and we felt OMG's improvements to the UML specification would lead to increased opportunities for automated UML-based testing. This assumption is somewhat confirmed by our analysis (Section IV). We selected conference and journal venues (available in our online technical report [3]) that, from our past experience as researchers in the domain, we knew would likely publish work on RBT and UML based testing. Since the automatic search was conducted in parallel with the manual search, venues that had multiple relevant papers appearing in the automatic search were added to the list of relevant venues for the manual search [3].

Automatic Search. We searched five digital libraries that were highly relevant to software engineering and have been recommended by others [5, 8]: IEEE Xplore, ACM Digital Library, SpringerLink, Scopus and Inspec, during the period 1990-2011. Based on our research questions, we formulated the initial query string (Q0):

```
Q0 "Requirement based testing" OR "Requirement Driven Testing" OR "Specification based testing" OR "Specification Driven Testing"
```

Given our focus on RBT, we also performed a survey (though not as systematic as a mapping study) of requirement modeling techniques [10] and the definition of the word requirement itself (see [3] for details). We identified techniques as varied as natural language specifications to diagrammatic notations that specify system behavior (e.g., state machine) or interaction scenarios (e.g., sequence diagram). Considering that we are interested in testing techniques to be used in a UML context, we selected the following requirement modeling/specification techniques: use case, sequence diagram, activity diagram, state machine diagram, natural language. We selected this subset of UML diagrams, and not other diagrams like timing diagrams though they could be used to specify requirements, since these diagrams have been shown to be the most used by practitioners [6, 12]. We thus formulated the following additional queries:

```
Q1  "use case" AND test*
Q2  "sequence diagram" AND test*
Q3  "activity diagram" AND test*
Q4  ("statechart" OR "state machine" OR "state
    diagram") AND test*
Q5  "natural language" AND requirement AND
    test*
```

We kept word "requirement" in Q5 since otherwise query `"natural language" AND test*` was returning too many false positives, e.g., RBT techniques in other engineering disciplines than software engineering. We added this query because use case descriptions are usually written in natural language.

Because we had an interest in safety critical and embedded real-time systems we added the following two queries as well (in these two queries we also kept word "requirement" for the same reasons as previously):

```
Q6  "safety critical" AND requirement AND
    test*
Q7  "embedded" AND "real time" AND requirement
    AND test*
```

As illustrated above, we followed Kithenham's recommendations [8] and identified terms as well as synonyms and alternative spellings that were specific to our research questions, and then used ANDs and ORs to construct sophisticated queries.

We searched the selected five digital libraries with those queries, accounting for the slightly different formats the search engines required. The search with Q0 was performed in October 2010 and the one with Q1-Q7 was performed in June 2011. The search was performed within the title, abstract and keywords of papers.

B. Article Selection: Inclusion/Rejection Criteria

Our article selection entailed several steps (Figure 1). During the initial, manual plus automated search, any paper discussing an approach related to some testing activities from software requirements was added to the set of relevant papers.

Throughout the manual search, after scanning the list of papers in a conference proceeding or a journal issue, we considered all the papers that had a title relevant to software verification and validation. We then selected the ones that discussed any kind of RBT technique by reading the abstracts, introduction and conclusion sections. In a few cases we had to use the full text of the paper to make the final decision. For each selected paper, we also examined the list of references to find potential additional relevant papers.

We followed a similar procedure during the automatic search. However, we had to strengthen the exclusion criteria. Whenever possible, we used the filters provided in digital libraries to limit the scope of search to computer engineering or related areas. But, we still obtained many papers from other disciplines like mechanical engineering or civil engineering. We also excluded those papers.

After removing duplicates (obtained from different databases, from both the manual and automated searches) we defined a more precise set of inclusion/exclusion criteria to select the final set of primary studies:

1. We excluded papers on testing based on design artifacts, such as testing from design patterns specified in a class diagram;
2. We excluded papers that discussed testing model transformations;
3. We excluded papers discussing concepts like test case prioritization, test case optimization, test effort estimation, test planning and coverage analysis;
4. When we had several papers describing one single approach by the same author(s), we only considered the most recent one and if available the most recent journal paper, assuming that we would then obtain the most complete description of the approach;
5. We only included papers that discussed testing based on the UML diagrams mentioned previously.

III. Results

The manual search resulted in a total of 147 papers, 51 of which are journal papers. Merging results of the manual and the automatic searches led to a set of 1,275 papers. After removing duplicates, we obtained a set of 702 papers. Applying our set of inclusion/exclusion criteria resulted in 100 unique papers as the final set of primary studies. The complete list of papers as well as descriptive results of automatic and manual searches are available in a technical report [3].

Table 1. Automatic Search (descriptive statistics)

	IEEE		**Inspec**		**Scopus**		**ACM**		**Springer**	
	T	**R**	**T**	**R**	**T**	**R**	**T**	**R**	**T**	**R**
Q0	64	49	123	86	133	87	26	11	20	12
Q1	124	38	290	83	589	93	46	12	52	22
Q2	30	19	61	38	154	74	20	10	8	6
Q3	25	21	45	33	98	46	20	13	6	3
Q4	405	130	NA	NA	NA	NA	21	7	114	37
Q5	38	10	194	38	89	15	18	4	14	3
Q6	75	12	286	42	209	26	20	3	21	3
Q7	113	9	399	16	189	10	21	2	21	4

IV. Analysis

Our analysis entails comparison (Section IV.A) and classification (Section IV.B). We also observed evolutions over time (Section IV.C).

A. Comparison

We have defined seven criteria to compare primary studies, based on our research questions, to help us to extract from each paper data that we are interested in.
Requirement Model: The UML diagram used to model requirements is one of the most influencing factors when selecting and comparing testing techniques.
Test Model: In some papers, test cases are extracted directly from the requirement model in which case the test model is the same as the requirement model. In other papers, the requirement model is transformed into an intermediate model, which is used to generate test cases.
Level of Automation: We have defined four levels of automation: manual, partially automated, fully automated, and automatable. A partially automated technique entails some steps that require human expertise (e.g., expertise in the testing technique, in the system under test, in the domain of the system under test) while a fully automated technique does not. Automatable means that some steps of the technique are described with enough clarity and precision (e.g., an algorithm in pseudo code) to be automated. Should those steps be automated, the technique would then become either fully automated or partially automated (depending on whether some steps still require human expertise). If a primary study does not indicate that any of the steps can be automated, and therefore, the evaluation discussed in the paper (if any) is manual, then the technique is said to be manual.
Testing Level: We use a well known classification of testing levels [2]: acceptance testing, system testing, integration testing, module testing and unit testing.
Selection Criteria: These are the criteria being used to derive test cases from the test model. (Note that we make a difference between a selection criterion—a criterion being used to create tests, and a coverage criterion—a criterion to evaluate the coverage of an existing test suite, since creating technology and tool support for the latter is usually simpler than for the former.) Also we consider whether the selection criteria are based on the requirement model or some intermediate model when the test model is not the requirement model. This is important since in the latter case the mapping between the model element of the test model being exercised and elements of the requirement model is not necessarily straightforward.
Empirical Evaluation Technique: Since empirical evaluation has become an important part of software

engineering research [20] and it is becoming more and more common to provide empirical data to support an idea, we compared primary studies according to the kind of empirical evaluation they provide. Different taxonomies of empirical evaluation techniques exist and we selected one that distinguishes between an experiment, a case study, and an example [22]. To distinguish between experiments and case studies we use the notion of a state variable [22]. In an experiment, the state variable can take on different values to recognize the differences between various situations, e.g., a controlled situation and the situation under investigation, whereas the state variable assumes only one value in a case study. On the other hand, an example covers only some parts of a technique.

Empirical Evaluation Material: We want to distinguish between papers that show application of the proposed approach in an industrial context from the ones that show application on a smaller application or fractions of it, or from the ones that illustrate application on a toy example.

Others, before us, have described criteria to compare software testing techniques. Neto and Travassos discuss criteria similar to ours but in the context of model-based testing [13, 14]. We report on a smaller amount of information in this paper since this is only a conference paper whereas they reported on the result of a complete literature review. More specifically, among the 18 different criteria they discuss, seven are similar to ours, two are not adequate to our situation (e.g., they consider whether a technique is structural or functional, whereas we only deal with the latter), and the remaining attributes will be used in a future publication. Our future publication will also compare the empirical results reported in primary studies: for instance, in the software testing community, researchers are typically interested in studying the cost (e.g., in terms of number of test cases generated) and effectiveness (at finding faults, either real or seeded). Our set of criteria and the one of Neto and Travassos are subsets of the larger, more extensive characterization schema for software testing techniques [21].

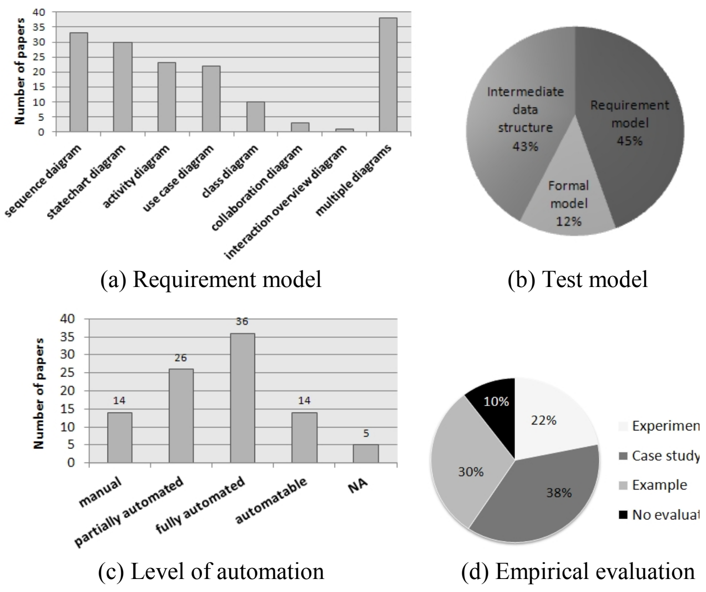
B. *Classification*

In this section, we classify the primary studies (papers) based on our seven comparison criteria.

Requirement Model. Figure 2 (a) summarizes the different UML diagrams that are used in primary studies. (We use only one term to refer to the UML 1.x and UML 2.x terminologies with respect to the state model.) Sequence and statechart diagrams have the highest number of hits. This is not surprising since these are probably the most used diagrams to specify behavior and interactions, and therefore functional requirements. Also, the statechart diagram is the most precisely defined UML diagram and therefore one of the most appropriate diagrams to derive tests from. We also noticed that 38 papers used more than one diagram as requirement model. This is a large amount and can be a result of the fact that usually more than one UML diagram is used during requirement engineering. The most common combination of models is use case diagram and either sequence diagram or activity diagram to model use case scenarios (17 papers). These 17 papers represent 77% of all the papers that rely on use cases for RBT. This is likely due to the fact that use cases and textual use case descriptions, although used a lot in requirement engineering, are not precise enough to be used alone during testing. Other common combinations are class diagram plus sequence diagram (six papers), sequence diagram plus statechart (four papers), and activity diagram plus statechart (three papers).

Test Model. 45% of the primary studies generate test cases directly from the requirement model: Figure 2 (b). The UML diagram that is the most used as a test model, either alone or in a combination with other diagrams, is the activity diagram (16 papers). The sequence diagram (13 papers) and statechart diagram (12 papers) come next. In 55% of the papers, the requirement model is transformed either into a formal model like Linear Transition System [17] or an intermediate data structure like communication tree [18] to generate test cases. This may be due to the lack of formality of UML or the desire to use a test model for which a testing technique already exists.

Level of Automation. Figure 2 (c). A lack of clear description for several steps of the proposed techniques



(a) Requirement model (b) Test model

(c) Level of automation (d) Empirical evaluation

Figure 2. Classification according to (a) the requirement model, (b) the test model, (c) the level of automation, and (d) the empirical evaluation

prevented us from classifying 5% of the primary studies. About 29% of the primary studies discuss automated tool support under the form of a single tool supporting several steps. These are a subset of the partially automated and fully automated primary studies. The rest of the (partially/fully) automated techniques discuss several pieces of tool support (but not a single bundle). Though not shown in Figure 2 (c), we note that all the techniques that use a formal model as test model are at least either partially automated or automatable. Equally interesting, 70% of the papers describing a manual process or that we were not able to classify are the ones that derive tests directly from the requirement model (i.e., the requirement model and the test model were the same), regardless of the kind or formal basis of requirement model used.

Testing Level. As expected, the majority of the primary studies (86%) describe a system testing technique. This is in accordance with the fact that system testing is designed to verify whether the assembled system meets its specification [2], and that requirements provide this specification. Other testing levels are integration testing (8%), unit testing (3%), and acceptance testing (2%). We also found one paper addressing non-functional testing (robustness testing). This small amount is not necessarily surprising since the diagrams we selected are more used to specify functional requirements than non-functional ones.

Selection Criteria. Since selection criteria depend on the test model, we do not classify primary studies based on this criterion: it would not be fair to say one technique does not use a criterion supported by another technique simply because they rely on different test models. However, we look at criteria used for similar test models. Some of the most used criteria for different diagrams are: state, transition for the statechart diagram; use case scenarios for the use case diagram; activity, transition, action for the activity diagram; message sequence path for the sequence diagram. We argue that, considering the extensive literature on test selection criteria, the criteria used in primary studies are among the simplest ones that exist. For instance, an activity diagram being very similar in structure (and purpose) to a control flow graph, it is surprising that other graph-based control- and data-flow criteria [2] are not experimented with. This also applies to sequence or statechart diagrams .

Four of the primary studies focus solely on empirical evaluation of different requirement based testing techniques either by means of case studies or experiments. We consider this to be a small proportion (4%) of empirical evaluations of different techniques, and this shows that there is room for more empirical studies in the area of RBT. Since these papers do not propose any new approach, we d not include them in the classification, but we have discussed them separately [3].

Empirical Evaluation. We found primary studies without any kind of empirical evaluation: Figure 2 (d). Only 22% of the primary studies have performed experiments to evaluate their approaches. This can be due to constraints in terms of length on what can be reported in a conference or workshop paper. Therefore we decided to analyze conference and workshop papers separately from journal papers and book chapters with respect to this comparison criterion (see figure 3). As expected experiments are used more often in journal papers while the most used empirical evaluation techniques in conference and workshop papers are case studies and examples. Note that we found two journal papers (out of 21) with no data on any empirical evaluation. The two journal venues are not listed in the Excellence in Research for Australia (ERA) journal and conference rankings; Nevertheless, the two papers were listed in the results of our automatic search.

Empirical Evaluation Material. The most occurring evaluation materials are toy example (34 papers), fraction of real world applications (18 papers), and industrial applications (13 papers). 16 studies include no discussion of the empirical evaluation material whatsoever: this includes the ten studies with no evaluation, indicating that six studies that report on an evaluation do not provide details on the experimental material.

C. Time Trends

Since the UML has evolved over time we speculated that our results would vary accordingly. Figure 4 (a) shows an increase in the number of primary studies around the year 2005. This can be caused by two factors. First, the more precise semantics of UML 2.0, released in 2005, might have made the UML more appropriate for testing. Second, 2006-2011 is the overlapping period between the manual and automatic searches. Therefore, the manual search could be the main cause of this increase. To better identify the root cause of the increase, we identified the papers found by only one or the other type of search and the papers found by both searches during the overlapping period 2006-2011: Figure 4 (b). The figure starts in 2005 to better present the trend for the automatic search. It shows that the trend is mostly due to the automated search.

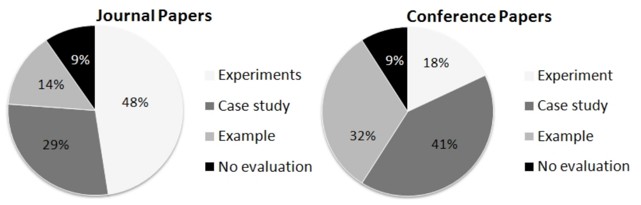


Figure 3. - Classification of journal and conference papers based on empirical evaluation method

We therefore conclude that the increase is due to the release of UML 2.0.

V. Threats to Validity

In any research work, there might be factors that can jeopardize validity. Our work is no exception and we need to discuss threats to validity. Systematic literature reviews and mapping studies are typically performed by a group composed of multiple researchers. In order to limit the threat of not having many researchers involved in the process, we followed guidelines [8] whereby we clearly defined and documented the important steps of the study including the review protocol, the research questions, the search strategy, the inclusion/exclusion criteria. These aspects are paramount to our study and results and this is the reason why we devoted a fair amount of space to describe them in this paper.

As mentioned before, both manual and automatic searches have advantages and drawbacks and we used both. One possible threat is that we might have missed a relevant (conference or journal) issue. However, our search was iterative and the dynamic search fed the manual one. Plus, the automatic search was not restricted in any way with that respect. Last, our knowledge of software testing research tells us that missing a venue is unlikely. A related threat is the possibility to miss a relevant paper from the identified venues because we manually selected them by only looking at the title, abstract, and keywords. However, when this information was not sufficient to make a decision we also looked at the introduction, the conclusion and the entire paper. This was, however, only necessary in a few cases, and we are confident we did not miss important papers. Another threat related to papers is the identification of primary studies. Although we tried to define a precise set of inclusion/exclusion criteria, there is always a chance of introducing a bias while applying them to different papers. Given the definition of those criteria, we however consider the risk is low. Note that we focused our study on UML-based RBT. We therefore excluded papers that describe testing techniques based on finite state machines (FSM) or extended finite state machines (EFSM). We do not consider this a threat, even though some of the (E)FSM techniques could be used in a UML context, since these techniques can be studied separately, there is a vibrant research community studying these techniques and there exists surveys describing them [9, 11].

The last threat to validity is about the classification of the primary studies. We tried to limit this threat by making our comparison criteria as precise as possible. We also reused existing classifications as much as possible. Nevertheless, there can always be cases for which the classification is affected by personal judgments (e.g., determining level of automation). In order to reduce the risks of introducing a bias in paper classification, 10% of the primary studies were selected randomly by the first author and submitted for classification to the second author. A few minor disagreements were noted and easily fixed.

VI. Conclusion and Future work

Requirement-based testing aims at starting testing-related activities as early as possible during the software development life cycle [1]. Since software testing is expensive, a great deal of research has been performed in the area of requirement based testing, with the hope to reduce testing costs. Unfortunately, as far as we know, to date, no systematic mapping study can help determine the current state of the art in requirement based testing.

The goal of our systematic mapping study was to identify published research works in the area of requirement-based software testing, and then provide a framework to evaluate them. Since the domain of requirement based testing is broad, such a mapping study is an important initial step before conducing more specific systematic literature reviews. In order to identify relevant papers we performed both an automatic search and a manual search, which resulted in about 1,300 papers. After removing duplicates and applying inclusion/exclusion criteria we obtained a set of 100 primary studies, which we compared by using seven complementary criteria.

The results of the classification lead us to make a number of interesting observations, including: (1) Almost all the primary studies (99%) discuss a functional testing technique. This indicates a lack of requirement-based testing approaches that address non-functional

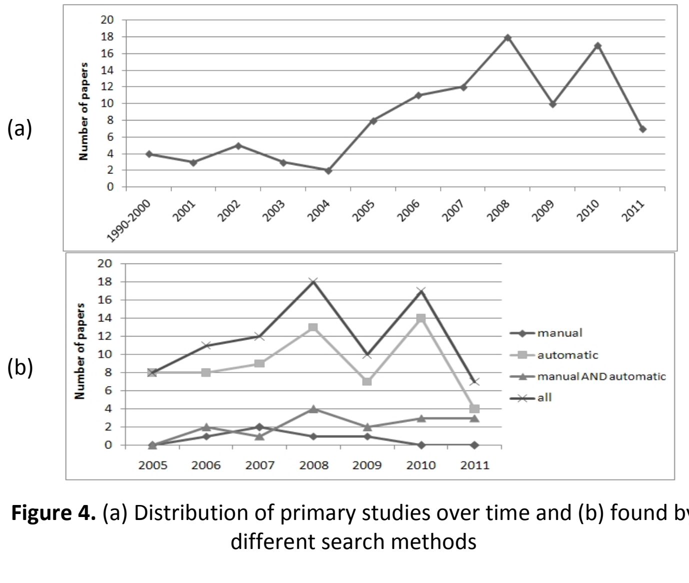


Figure 4. (a) Distribution of primary studies over time and (b) found by different search methods

requirements. This is however not entirely surprising since other UML diagrams than the ones we studied might be needed to specify and therefore test non-functional requirements; (2) Only a very small proportion of the primary studies (4%) evaluated different requirement-based testing techniques, suggesting a need for more empirical comparisons of requirement-based testing techniques; (3) A relatively small percentage of the primary studies (22%) provide a formal evaluation of the proposed testing technique under the form of an experiment. When only looking at primary studies published in journal venues, the proportion is higher (48%), though we observed that 23% of the journal papers only evaluate the proposed testing technique with an example or provide no evaluation; (4) The test selection criteria being used in the proposed requirement-based testing techniques seem to be among the simplest that have been suggested in the testing literature. For instance, we did not find a single primary study where data flow criteria are used, although data flow selection criteria can be used in sequence, activity, or state machine diagrams and they are known to complement control flow criteria.

Our future work includes an analysis of the primary studies in a qualitative way to answer questions like: What limits the application of the proposed techniques? What are the avenues for further research in UML-based, requirement-based software testing? We also plan to extend the scope of the review to other requirement modeling techniques than ones based on UML.

VII. Acknowledgment

This work was performed under the umbrella of a NSERC-CRD grant. The authors would like to thank NSERC, CRIAQ, CAE, CMC Electronics, and Mannarino Systems & Software for their financial support.

VIII. References

[1] G. Amit and B. Rajesh, "Testing functional requirements using B model specifications," Software Engineering Notes, 35, 2010, pp. 1-7.

[2] P. Ammann and J. Offutt, Introduction to Software Testing, Cambridge University Press, 2008.

[3] N. Asoudeh and Y. Labiche, "Requirement based software testing in a UML context: A systematic literature review", Tech. Rep. SCE-108,2011. Avialable from ,squall.sce.carleton.ca/ pubs/ tech_report /TR_SCE-11-08.pdf [retrieved: October, 2012]

[4] B. Beizer, Software Testing Techniques, International Thomson Computer Press, 1990.

[5] P. Brereton, A. B. Kitchenham, D. Budgen, M. Turner ,and M. Khalil, "Lessons from applying the systematic literaturereview process within the software engineering domain", JSS, vol. 80, 2007, pp. 571-583.

[6] B. Dobing and J. Parsons, "How UML is used," Com. of the ACM, vol. 49, 2006, pp. 109-113.

[7] C. Jones, Software Engineering Best Practices: Lessons from Successful Projects in the Top Companies, McGraw-Hill, 2009.

[8] B. A. Kitchenham, "Guidelines for performing systematic literature reviews in software engineering," Tech. Rep. EBSE-2007-001, 2007.

[9] R. Lai, "A survey of communication protocol testing," JSS, vol. 62, 2002, pp. 21-46.

[10] A. V. Lamsweered, Requirements Engineering, Wiley, 2009.

[11] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines-a survey," Proc. of the IEEE, vol. 84, 1996, pp.1090-1123.

[12] F. J. Lucas, F. Molina, and A. Toval, "A systematic review of UML model consistency management," IST, vol. 51, 2009, pp. 1631-1645.

[13] A. Neto and G.H. Travassos, "Evaluation of model-based testing techniques selection approaches: An external replication," Proc. ESEM, 2009, pp. 267-278.

[14] A. Neto and G.H. Travassos, "Model-based Testing Approaches Selection for Software Projects," IST, vol. 51, 2009, pp. 1487-1504.

[15] T. Pender, UML Bible, Wiley, 2003.

[16] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," Proc. EASE, 2008, pp. 68-77.

[17] S. Pickin, C. Jard, T. Jeron, J.M. Jezequel, and Y. Le Traon, "Test synthesis from UML models of distributed software," IEEE TSE, vol. 33, 2007, pp. 252-269.

[18] A. Pretschner, "Model-based testing in practice," Proc. Formal Methods, 2005, pp. 537-541.

[19] P. Samuel, R. Mall, and P. Kanth, "Automatic test case generation from UML communication diagrams," IST, vol. 49, 2007, pp. 158-171.

[20] F. Shull, J. Singer, and D.I.K. Sjoberg, Guide to Advanced Empirical Software Engineering, Springer, 2008.

[21] S. Vegas and V. Basili, "A Characterization Schema for Software Testing Techniques," ESE, vol. 10, 2005, pp. 437-466.

[22] C. Wholin, P. Runsen, M. Host, M. C. Ohlsson, B. Rengell, and A. Wessslen, Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, 2000.

[23] RTCA: Software Considerations in Airbone Systems and Equipment Certification. Radio Technical Commission for Aeronautics (RTCA), Standard Document no. DO-178C. (2011)