Science of Computer Programming 95 (2014) 275–297

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

# NAT2TEST$_{SCR}$: Test case generation from natural language requirements based on SCR specifications

Gustavo Carvalho[a,*], Diogo Falcão[a], Flávia Barros[a], Augusto Sampaio[a], Alexandre Mota[a], Leonardo Motta[b], Mark Blackburn[c]

[a] UFPE – Centro de Informática, 50740-560, PE, Brazil
[b] Embraer, 12227-901, SP, Brazil
[c] Stevens Institute, 07030-5991, NJ, USA

## HIGHLIGHTS

- We generate test cases for reactive systems from natural language requirements.
- Requirements are semantically analyzed using case frames and translated into SCR.
- SCR is used as a hidden formalism to generate test cases using the T-VEC tool.
- The strategy is evaluated using examples from the Aerospace/Automotive industry.
- We outperformed random testing (Randoop) concerning performance and mutation score.

## ARTICLE INFO

*Article history:*
Received 20 July 2013
Received in revised form 6 June 2014
Accepted 10 June 2014
Available online 16 June 2014

*Keywords:*
Requirements
Test generation
Natural language processing
Software cost reduction
SCR

## ABSTRACT

Formal models are increasingly being used as input for automated test generation strategies. Software Cost Reduction (SCR), for example, was designed to detect and correct errors during the requirements phase, also allowing test generation. However, the syntax of SCR and other formalisms are not trivial for non-experts. In this work, we present a strategy for test case generation from natural language requirements that uses SCR as an intermediate and hidden formalism. To minimize textual ambiguity, requirements are written according to a controlled natural language. Syntactically valid requirements are mapped into their semantic representation using case frames, from which SCR specifications are derived. These specifications are then used by the T-VEC tool to generate tests cases. Our strategy was evaluated in four different domains: (i) a vending machine (toy example); (ii) a control system for safety injection in a nuclear power plant (publicly available), (iii) one example provided by our industrial partner Embraer; and (iv) the turn indicator system of Mercedes vehicles (publicly available). As a baseline we considered random testing, and, in general, our strategy outperformed it in terms of performance and mutant-based strength analysis.

* Corresponding author.
*E-mail addresses:* ghpc@cin.ufpe.br (G. Carvalho), dfdf@cin.ufpe.br (D. Falcão), fab@cin.ufpe.br (F. Barros), acas@cin.ufpe.br (A. Sampaio), acm@cin.ufpe.br (A. Mota), leonardo.motta@embraer.com.br (L. Motta), mark.blackburn@stevens.edu (M. Blackburn).

http://dx.doi.org/10.1016/j.scico.2014.06.007
0167-6423/© 2014 Elsevier B.V. All rights reserved.

## 1. Introduction

During the last fifty years, there has been a significant increase of embedded HW-SW components in critical systems. A report from NASA [1] highlights that, from 1960 to 2000, the amount of functionalities provided to military aircrafts by embedded software has grown from 8% to 80%. This scenario is not restricted to the aerospace industry. The automotive industry, for instance, has become even more dependent on embedded components. According to results reported in [2], in 2009 some cars already summed up 100 million lines of code, and this number can reach 200 or 300 million lines of code within a few years. Clearly, this trend increases software size and complexity, and strongly impacts safety and reliability of critical systems.

Currently, many researches are focusing on how to achieve the safety levels required for critical systems. One approach to deal with this scenario relies on formal verification [3,4]. Although formal verification, in theory, guarantees absence of errors, this goal is not generally feasible for large and complex systems, whose required processing power and memory typically make the verification intractable.

An alternative and complementary approach that usually scales better than formal verification is Model-Based Testing (MBT) [5,6]. MBT aims to provide for a more agile, less expensive, and better quality, testing process [7]. These benefits are usually obtained through the automatic generation (and execution) of test cases, besides the automatic generation of test data, from models of the specification. However, it is important to bear in mind that in complex and real situations a complete automated process may not be feasible, and thus a semi-automated testing process is more usual.

In the context of MBT, the quality of the specification models is crucial for an effective testing campaign. For instance, if the model does not properly capture the system behavior, the approach may generate incorrect and inconsistent tests. Thus, it is desirable to describe the expected system behavior using some (semi-)formal notation, which may consider different abstraction levels. Examples of such notations are the Unified Modeling Language (UML) [5], Lustre [8], the Software Cost Reduction (SCR) [9], process algebras (such as *Communicating Sequential Processes* – CSP [10]) and Labelled Transition Systems (LTS) [11], among others. The fundamental goal is indeed to avoid inconsistent and incomplete specifications.

However, despite the benefits provided by MBT, the demand for specifying the desired system behavior using (semi-) formal models may become an obstacle for its use. In 2009, the Federal Aviation Administration (FAA) published a report [12] which discusses current practices concerning requirements engineering management. The report states that "*... the overwhelming majority of the survey respondents indicated that requirements are being captured as English text, shall statements, or as tables and diagrams ...*" ([12], p. 64). This finding suggests that the use of MBT techniques in industrial contexts may require specialists (usually mathematicians, logicians or computer scientists/engineers). Yet, most of these models are not available in the very beginning of the system development project. In the initial phases, usually only high-level requirement descriptions are available. Therefore, the use of MBT tends to be postponed to later development phases. To overcome these drawbacks, we claim that Natural Language Processing (NLP) [13] can be used to support MBT.

### *1.1. NAT2TEST$_{SCR}$ – proposed approach*

We present in this work NAT2TEST$_{SCR}$, a strategy for the automatic generation of test cases from requirements written in natural language. This way, we dispense with a specialist on (semi-)formal languages, still allowing early use of MBT. In this work, the formal SCR models used by the NAT2TEST$_{SCR}$ strategy are automatically generated from the natural language requirements, and are hidden from the user of the strategy. The strategy NAT2TEST$_{SCR}$ can in fact be considered a specialization of a more general strategy (NAT2TEST) that would be independent of the formal notation used as well as the adopted test generation and execution approach. Herein, we focus on its specialization using the SCR formalism.

Aiming to avoid textual ambiguity or misinterpretations, in this work requirements are written according to a Controlled Natural Language (CNL), named *SysReq-CNL*, which is simple to understand and easy to use. Furthermore, it imposes standardization on requirements without losing naturalness.

A complete NLP system is usually composed of five processing levels, depending on its aim: morphological analysis, syntactic analysis, semantic mapping, discourse analysis and pragmatic analysis. Concerning MBT, the works reported in [14–16] address the joint use of MBT and NLP based on the second and third NLP levels. Our work, similarly, considers these two NLP levels in separate processing phases. We do not perform discourse and pragmatic analysis, since we are dealing with system requirements (rather than with discourse/dialogues).

The NAT2TEST$_{SCR}$ strategy comprises three processing phases: syntactic analysis, semantic analysis, and SCR generation. The first phase (*syntactic analysis*) comprises a morphosyntactic (morphological and syntactic) analysis of the input requirements, in order to generate its corresponding syntax tree. Afterwards, the second phase (*semantic analysis*) maps the syntax trees into a semantic representation based on the case grammar theory [17]. Finally, the last phase (*SCR generation*) delivers an SCR specification, which can be used by test generation tools to deliver test cases. This way, we can increase quality without needing a specialist on formal languages like SCR.

Our approach is tailored to a particular type of critical systems: data-flow reactive systems. According to [18], a data-flow reactive system interacts cyclically with its environment by means of an input and output set of events. These events are modeled as input (monitored) and output (controlled) variables that obey some timing constraints. Changes of the input variables trigger particular changes of the output variables.