Test Scenario Generation for Feature-Based Context-Oriented Software Systems

Pierre Martou
ICTEAM, UCLouvain
Louvain-la-Neuve, Belgium
pierre.martou@uclouvain.be

Kim Mens
ICTEAM, UCLouvain
Louvain-la-Neuve, Belgium
kim.mens@uclouvain.be

Benoît Duhoux
ICTEAM, UCLouvain
Louvain-la-Neuve, Belgium
benoit.duhoux@uclouvain.be

Axel Legay
ICTEAM, UCLouvain
Louvain-la-Neuve, Belgium
axel.legay@uclouvain.be

This extended abstract is based on a March 2023 publication in the Journal of Systems and Software (https://dl.acm.org/doi/abs/10.1016/j.jss.2022.111570). It appeared in a Special Section on *Systems and Software Product Lines of the Future* and was submitted through the SPLC Special Issue call following SPLC'22. These results are novel and not an extension of any previous publication in SPLC (including SPLC'22). As such, they have not been presented at a Software Product Lines Conference before.

1 EXTENDED ABSTRACT

Feature-based context-oriented programming reconciles ideas from context-oriented programming, feature modelling, and dynamic software product lines. As a result, feature-based context-oriented programming (FBCOP) offers a programming language, architecture, tools, and methodology to develop context-aware software systems. Such systems use contextual information identified in their immediate environment to select and activate appropriate behaviour at runtime. How to test features-based context-oriented software systems is critical to this work and provides the subject of our research. Systems developed using FBCOP represent contextual information via a context model and behaviour via a feature model (using feature modelling). A mapping model details which contexts trigger which features. Testing FBCOP systems is challenging given the exponential number of possible configurations. Our first research question is thus **How to generate a pertinent yet tractable set of test scenarios for a given FBCOP application?** (RQ1). To respond, we propose an approach that exploits feature-based context-oriented modelling for efficient test sampling. We use *combinatorial interaction testing (CIT)* to create small but relevant test suites that cover all valid pairs of contexts and features. We accomplish this through an efficient greedy generation algorithm with SAT solving. CIT testing has logarithmic growth of the generated test suites sizes according to the number of features. We demonstrate that adding a context model does not increase the theoretical size of generated test suites compared to feature-based systems (without knowledge of the environment).

Human effort needs to be invested to use generated test suites. Minimising this effort leads to our second research question: **How to minimise the effort of simulating these generated test scenarios?** (RQ2). The number of context switches involved in test configurations in a test suite represents the required effort to simulate (or reconfigure) the system being tested. More switches between test configurations can suggest longer execution times due to feature (un)deployment, increased difficulties in locating a specific error that occurs between largely different configurations, worse maintenance of more complex test suites, etc. For these reasons, we aim to minimise the number of context switches or the *creation cost* in a test suite. We propose a rearrangement algorithm to accomplish this goal. Rearranging the test configurations within a test suite results in a reduction of 45% in creation cost (43% in the number of feature switches) compared to no rearrangement.

Next we evaluate our testing approach in practice: **How effective is our approach in finding errors?** and **What type of errors can we find using this testing approach?** (RQ3a, RQ3b). To find out, we first illustrate our testing approach in a case study and write usage scenarios directly inspired by a generated test suite. This highlights certain design issues, e.g., two errors in the context-feature mapping, an incoherent feature adaptation, incorrect modelling of the environment, etc. This approach also shows the usefulness of creation cost minimisation; fewer context and feature switches in a test suite lead to shorter usage scenarios that are easier to write and analyse. To answer RQ3a, RQ3b, and RQ4 "**How relevant and easy-to-use is the testing approach for developers?**" in more detail we conducted a study with several developers who designed, developed, and tested a small feature-based context-oriented application using our approach. While this study reported good effectiveness and a wide range of design errors, without a good test oracle the developers encountered difficulties to use the generated test suites for implementation testing.

Context-aware systems are subject to evolution (new sensors, new environments, expansion of system features, etc). This begs the question: **How to incrementally adapt a previously generated set of scenarios upon the evolution of the application to a new version?** (RQ5). Reusing a previously generated test suite has the same goal as test rearrangement: reducing the test effort with our approach. We propose an algorithm and two strategies for reuse; our work achieves 75% reduction in the creation cost of a test suite for an updated system when reusing a previously generated test suite, as compared to generating an entirely new test suite.

SPLC'23, August 28–September 1, 2023, Tokyo, Japan
2023. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn