Automation of Test Case Assessment in SPLs — Experiences and Open Questions

Josh Taylor[1], Alexander Knapp[2],
Markus Roggenbach[1], Bernd-Holger Schlingloff[3]

[1] Swansea University, Wales, United Kingdom
joshdt001@gmail.com
[2] Universität Augsburg, Germany
[3] Humboldt Universität and Fraunhofer FOKUS, Germany

Abstract. This research idea turns a theory for test case assessment in the model-based development of multi-variant systems, so called Software Products Lines (SPL), into practice. To this end, we provide a tool chain for automated test case assessment, validate it on the example of a coffee machine product line, and finally, successfully apply it to "The Body Comfort System" product line from the automotive domain.

1 Introduction

The concept of a software product line originates by the work of D. Parnas [9]. It has gained much attention by the research and consultancy of the Carnegie Mellon University Software Engineering Institute [1,7]. According to the CMU-SEI definition, "a software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way"[4] . SPLs are abundant in today's software-intensive systems: most electronic control units, e.g., in cars or trains, come in multiple variants, as well as consumer products such as coffee machines, dishwashers, mobile phones, etc.

A challenge common to the development of these systems, say, an automotive body comfort systems, is that their built-in software is similar, but not identical in all products; there are slight differences according to the features exhibited by a particular product. Sources of variability include planned diversity for different user groups, evolution and enhancement of products, and re-use of modules from one product in another one. SPL engineering addresses this challenge. The main goal of SPL development is the strategic re-use of software artefacts. There have been various approaches to re-use: by copy and paste, macros, subroutines, modules, objects, components and services. The common problem in all of these approaches is that re-use increases the probability of errors. Therefore, quality assurance for SPLs is of utmost importance.

Given an SPL and a software test suite, not all tests apply to all possible products. Often, it is clear from the context whether a test can be executed with a product or

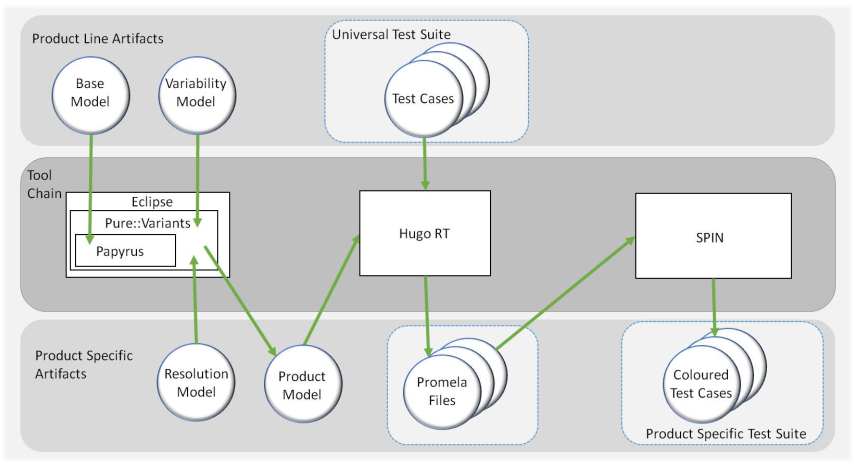
[4] CMU Software Engineering Institute: Product Line Web Page. http://www.sei.cmu.edu/productlines/ (2015/01/11)

not. However, there are cases where this is not obvious: consider the case that a certain additional feature blocks some behaviour present in the 'standard' product. In general, for a sufficiently expressive specification language the problem of test case assessment is undecidable.

In [4,5], some of the authors presented a theory for test case assessment in the model-based development of multi-variant systems. This deals with both positive (green) and negative (red) test cases, and introduce a third colour (yellow) for test cases whose outcome is not determined with a given product model. This means that it is needless to execute them with products based on this model. This approach thus allows to assess and select those test cases from a universal test suite which are relevant for a given product.

2 Tool Chain

Our tool chain can be separated into two main parts: the modelling and the test colouring part. It re-uses existing tools, where our contribution lays in the automation of their co-operation.



For the modelling part, we completely rely on existing technology. We utilise the UML model editor PAPYRUS[5] to define a base model (consisting of class diagrams, state machines, and a composite structure diagram) and PUREVARIANTS[6] to deal with variability, namely to create a feature model and a variability model (above shown as one integrated variability model). In PUREVARIANTS, the user then realises a resolution model by selecting which features to include. This results in a product model in UML.

The test colouring uses two tools, namely HUGO/RT[7] and the SPIN model-checker[8]. Here, we extended HUGO/RT to automatically take a list of test cases and a product model in order to translate these into executable PROMELA code, which can be checked

[5] `www.eclipse.org/papyrus/`(2016/08)
[6] `www.pure-systems.com/products/pure-variants-9.html`(2015/06)
[7] `www.informatik.uni-augsburg.de/en/chairs/swt/sse/hugort/`(2016/08)
[8] `spinroot.com/spin/whatispin.html`(2016/08)

by SPIN. Each applicable test case results in a separate Promela file. This file is then processed with SPIN in up to three passes for different reachability properties of the encoded automaton.

2.1 Validation: Coffee Machine

In [4,5], some of the authors presented a small Software Product Line of an automated coffee machine. A number of test cases were presented, and manually coloured against various products. Our tool chain was able to produce all test colourings as predicted – except one, where it turned out after careful analysis that the manual process, due to human error, had resulted in a wrong colouring. Overall, this result increases trust in our tool chain. Furthermore, it demonstrates the need for tools: even for a relatively simple test case human error occurred.

3 An Industrial Size Case Study: Body Comfort System

As a case study of industrial size, we have chosen the “Body Comfort System Case Study” developed by Lity et al. [6] in order to demonstrate their delta orientated SPL test method. The system is made up of a mandatory interface, a mandatory door system and an optional security component.

Lity et al. give in total 64 test cases represented as Message Sequence Charts. We have represented a significant subset of these test cases in our approach, i.e., we encoded them as sequences of occurrences and dispatches of UML events and were running them through our tool chain in order to assess their colour. This resulted in having multiple test cases for each product model. Here, we studied all 18 product models considered in [6].

Looking at each product model, the first question was if a test case was applicable, i.e., we checked if the alphabet of events of the test cases was a subset of the alphabet of the product model. Here it turned out that the notion of applicability in Lity et al. is the same as ours. Furthermore, in the average, only about 1/3 of the test cases are applicable to the product models. If a test was applicable, we were running our test colouring procedure.

Concerning time, the colouring time depends on a number of different aspects. Time is growing with the length of a test case: short test cases consisting of 3 or 4 events take 1–2 seconds, while longer ones of about 20 events can take 3–4 minutes. Also, time grows with the number of different state machines involved within the test case. Finally, time grows with the number of transitions in these state machines. Overall, colouring time is below 10 minutes per test case. In this respect, we consider our approach to be practically feasible, as most test cases can be coloured fast, and about two thirds of them are excluded due to the basic applicability criterion.

Concerning colouring, it turns out that all test cases from [6] are actually green ones, i.e., they express desired behaviour. It does not come as a surprise that there are no yellow test cases: the UML models are close to implementation where all relevant design decisions have been resolved. That there are no red test cases is a question of methodology: apparently, Lity et al. were concentrating on demonstrating expected behaviour rather trying to demonstrate that certain error situations have been avoided.

4 Conclusions and Open Questions

We have successfully implemented a tool chain for SPL test assessment according to the theory presented in [4,5]. We validated the tool chain on a simple example, and showed that the developed implementation scales up. However, there are a number of research questions arising:

1. What are appropriate coverage metrics for SPL models and coloured test cases?
2. Currently we are focusing on individual products for colouring. However, in principle, it should be possible to colour classes of products. This would reduce colouring time. How to refine the base model adequately?
3. As we obtain only green test cases in our case study, we presume that Lity et al. were performing testing for functionality. How to extend this to testing for safety, i.e., "show whether or not each software module performs its intended function and does not perform unintended functions" (IEC 61508)? A first step towards this would be to formulate safety properties as test objectives, that one then would turn into red test cases.

Overall, it is still an open question of how to relate our approach with incremental development methods such as step-wise refinement and software evolution.

References

1. P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.
2. CMU Software Engineering Institute: Product Line Web Page. `http://www.sei.cmu.edu/productlines/`$^{(2015/01/11)}$.
3. Hugo/RT. `http://www.pst.informatik.uni-muenchen.de/projekte/hugo/`$^{(2015/June)}$.
4. A. Knapp, M. Roggenbach, and B.-H. Schlingloff. On the use of test cases in model-based software product line development. In S. Gnesi, A. Fantechi, P. Heymans, J. Rubin, K. Czarnecki, and D. Dhungana, editors, *18th International Software Product Line Conference, SPLC '14*, pages 247–251. ACM, 2014.
5. A. Knapp, M. Roggenbach, and B.-H. Schlingloff. Automating test case selection in model-based software product line development. *International Journal of Software and Informatics*, 9:153–175, 2015.
6. S. Lity, R. Lachmann, M. Lochau, and I. Schaefer. Delta-oriented software product line test models – The Body Comfort System Case Study, 2014. Informatik-Bericht Nr. 2012-07, TECHNISCHE UNIVERSITAT CAROLO-WILHELMINA ZU BRAUNSCHWEIG.
7. J. D. McGregor, L. M. Northrop, S. Jarrad, and K. Pohl. Initiating Software Product Lines. *IEEE Softw.*, 19(4):24–27, 2002.
8. Papyrus, Eclipse Model Development Tools webpage. `http://www.eclipse.org/papyrus/`$^{(2016/08/9)}$.
9. D. L. Parnas. On the Design and Development of Program Families. *IEEE Trans. Softw. Eng.*, 2(1):1–9, 1976.
10. Promela language reference. `http://spinroot.com/spin/Man/promela.html`$^{(2015/June)}$.
11. Pure::Variants. `https://www.pure-systems.com/products/pure-variants-9.html`$^{(2015/June)}$.
12. Spin reference page. `http://spinroot.com/spin/whatispin.html`$^{(2015/June)}$.