Efficient and Effective Testing of Automotive Software Product Lines

Anastasia Cmyrev
Daimler AG, Benz-Str., 71063 Sindelfingen, Germany

Ralf Reissing*
Coburg University of Applied Sciences and Arts, Friedrich-Streib-Str. 2, 96450 Coburg, Germany

* Corresponding author. E-mail: ralf.reissing@hs-coburg.de
Received: 18 April 2014; Accepted: 8 May 2014; Published online: 20 May 2014
DOI: 10.14416/j.ijast.2014.05.001

Abstract
Within the automotive industry, the clients' high demand for individually customized products results in a growing number of product variants. In order to control the complexity of developing these variants, a product line approach is used that supports reuse of the common set of assets (e.g. requirements, software code, and test cases). The naive approach to assure quality of the variants derived from the product line would be to individually test each variant. However, due to the large number of variants it is virtually impossible to test all variants in detail and still release the product line on time. In this paper, we propose a new test approach based on requirements coverage and variant properties coverage (i.e. feature coverage) that leads to effective but also efficient test coverage of all variants of the product line. A set of variants is selected that is 1) as small as possible, 2) covers all requirements of the product line, and 3) covers all features of the product line. This small set of variants is then tested in detail. Reducing the number of tested variants allows for deeper testing and thus finding more defects. Because of the coverage achieved by the variant set, the quality of all other variants can be inferred from the test results. Finding the optimal set of variants in itself is a very hard problem, i.e. in an industry setting with huge numbers of possible variants it is practicably infeasible to strive for the optimum. Therefore, we developed two approaches which are not guaranteed to find the optimal set but get very near to it in little time. First, a greedy algorithm was created which produced very good results in very little time in all case studies. For example, for a system with one million variants a set of eight representing variants was selected within seconds. Second, a simulated annealing approach was evaluated in order to check for further potential of improvement. However, the case studies showed that the greedy algorithm is the better choice for practical results.

Keywords: *Product line testing, Requirements coverage, Variant coverage, Automotive embedded systems*

1 Introduction

Within the automotive industry, individual customers' demands, market specific requirements, technical variability, strategic decisions etc. cause the number of features and options to grow. The resulting space of valid vehicle variants increases exponentially with the number of possible options [1]. For example, the current Mercedes-Benz A-class allows customers to configure approx. 10^{15} possible variants. The methodology of Product Line Engineering (PLE) helps to cope with the resulting complexity and variability of these systems in the development process. PLE supports a systematic and proactive reuse of development artefacts by taking advantage of the products' common set of assets and hence efficiently reduces the time-to-market and the development costs of the product line while at the same time improving its quality [2].

The PLE approach chosen at Daimler AG is to use common feature models for all development artefacts [3]. In this paper, the focus is on Product Line (PL) requirements specifications and the associated test cases in test specifications. Such generic specification documents allow deriving specific product specifications by selecting the relevant features with

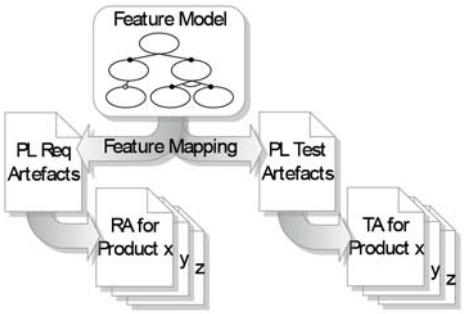

Figure 1: Overview of the relationships between feature model, requirements artefacts (RA) and test artefacts (TA) at Daimler AG.

respect to a certain product, as illustrated in Figure 1. This way, the specifications have a generic character and provide a high degree of reusability for the development artefacts [4].

Before products derived from a product line can be delivered to customers, they need to be tested whether they work as intended. This would mean running all test cases associated with all products planned for release to customers [5]. As the number of products is very high, this is infeasible as testing would take much too long. In order to reduce the number of test cases executed, there are two vectors of approach which can be combined. First, the number of products tested can be reduced to a subset representing all products. Second, the number of test cases executed over all products is reduced, e.g. using appropriate regression strategies for common features. In this paper, the main focus is on the first approach, i.e. selecting a subset of products as small as possible that still gives sufficient confidence that products tested as well as products not tested will have the necessary quality for release.

2 Related Work

In the literature, three main approaches for selection of variants for product line testing can be identified. First, the combinatorial testing, which determines variants based on the interaction coverage of features, e.g. pairwise coverage [6]. The approach exclusively uses information from the variability model but achieves an effective reduction of the product subset for testing. The second strategy prioritizes variants according to criteria like most critical, most sold etc. [7]. This strategy requires a lot of effort for providing the necessary additional information. The third method is similar to the one used in this paper and selects variants based on requirements and architecture coverage [1]. The drawback is that it does not scale with real world product lines.

3 Approach

In this paper, the approach to selecting a small but sufficient subset of variants maximizes both requirements coverage and feature coverage. Requirements coverage is defined as the ratio of requirements in the product line requirements specification which are present in at least one variant selected to the total number of requirements. 100% requirements coverage makes sure that there are no untested requirements. Feature coverage is the ratio of all features of the common feature model present in at least one variant selected to the total number of features that were actually used in the requirements specification. In general, feature coverage is largely achieved by variants selected for requirements coverage already, but may add variants to cover special cases. 100% feature coverage makes sure no feature present in any product is untested.

Feature coverage makes sure that all features of the product line are taken into account. Requirements describe the behavior of the product line and also reflect relationships between features (e.g. a requirement can be associated with various features). For testing, it is important to consider both requirements and features.

Finding the minimal set of variants that yields 100% requirements and feature coverage is a very hard problem: the time needed grows with the increasing number of variants, which in general grow exponentially with the number of features in the feature model. In practical applications the number of variants is huge, therefore fast methods for selection are needed that result in subsets small enough but possibly not minimal. We evaluated both greedy algorithms (a local search approach) and simulated annealing (a global search approach).

3.1 *Greedy algorithm*

Greedy algorithms [8] work towards an optimal solution by an iterative approach. In each iteration, from all possible steps towards a solution the one step is chosen

that gets most close to the optimum. This requires a measure for the quality of intermediate results. Here, this is the combination of requirements and feature coverage.

Some requirements may be relevant for exactly one variant, therefore these variants need to be selected in any case. Therefore, in a first step, the set of selected variants is initialized with such variants. Interestingly, in both case studies (see below) there were not such variants.

In the main loop, from the remaining variants the one is selected that yields the biggest improvement in requirements coverage. If there are several candidates with the same yield, among those the one with the biggest yield in feature coverage is chosen. If there is still more than one candidate left, the choice is made random among them. The loop is iterated until both requirements coverage and feature coverage reach 100%. The algorithm is described in detail in [9].

For the average case the run time of the Greedy Algorithm is $O(|C| \cdot \log(|C|))$. It is mainly determined by the number of configurations C which have to be sorted according to their coverage of requirements and features within a priority queue.

The memory consumption mostly depends on the calculation of the configurations C from the feature model and which requirements R and features F are valid for which configurations: $O(|C| \cdot |R|+|C| \cdot |F|)$. The calculations of these lists are performed before the actual selection in the Greedy Algorithm.

3.2 *Simulated annealing*

Simulated Annealing (SA) is a method for a random-based search for a global optimum [10]. While a Greedy Algorithm only accepts intermediate steps that increase the quality of the solution, SA accepts worse solutions with a certain probability that decreases over iterations until it reaches zero. The big advantage is that SA may get out of local optima in the solution space, which a Greedy Algorithm cannot.

Here, a random subset of variants of a fixed size is created. Each intermediate step swaps a random variant in the subset with another random one from the rest of the variants. The quality metric E of the new subset $C_{selected}$ is defined by the ratio of uncovered requirements R and features F, multiplying them with individual weights w and adding the results to the weighted ratio of selected variants to all variants C_{all} (see eq. 1). E needs to be minimized for best results.

$$E = w_r \frac{|R_{uncover}|}{|R_{all}|} + w_f \frac{|F_{uncover}|}{|F_{all}|} + w_c \frac{|C_{selected}|}{|C_{all}|} \quad (1)$$

with $w_r = \frac{127}{210}, w_f = \frac{80}{210}, w_c = \frac{3}{210}$.

Better new subsets are always accepted. Worse new subsets are also accepted with a certain probability which gets lower by each iteration. Else the former solution is retained and used in the next iteration.

The acceptance probability depends on a simulated cooling schedule in analogy to annealing in metallurgy. The cooling is determined by $T := \frac{T}{1+\beta T}$, with a starting temperature T_{init} = 100°C and an end temperature T_{end} = 0.001°C, which is also used as the stop criterion for SA. The cooling rate β determines how many iterations are executed between T_{init} and T_{end} to search for a better solution. In the case studies values between $0.0005 \leq \beta \leq 0.1$ were evaluated. The results of section 4.2 depict case studies with $\beta = 0.01$.

Using a fixed size of the subset is a constraint chosen from experience. First, adding or removing variants was also a valid change in an iteration, but this led to inferior results. Thus it was decided to use the size N of the subset yielded by the Greedy Algorithm as a starting size. SA runs were started with subset sizes from {N-b, …, N, …, N+d} with a small b and d, e.g. b, d = 3. The best solution of all subset sizes is being saved as the global best solution. As the size of the subset is considered in the last term of eq. 1, all other terms equal the smaller subset is better.

The run time of SA depends most on the number of iterations performed that swap variants times the number of subsets with fixed sizes: $O(\beta^{-1} T_{min}^{-1} \cdot (b+d+1))$. The memory consumption for SA is the same as for the Greedy Algorithm: $O(|C| \cdot |R|+|C| \cdot |F|)$.

3.3 *Test case selection*

Independent of the method used to select the subset of variants for testing, the test cases to be executed for each variant from the subset can be derived from the generic test specification of the product line by generating the specific test specification for each variant selected.

However, there may still be room for improvement

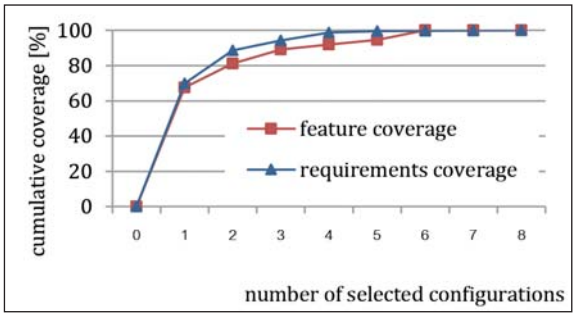

Figure 2: Greedy Algorithm: Cumulative coverages of selected configurations for Case Study 1.

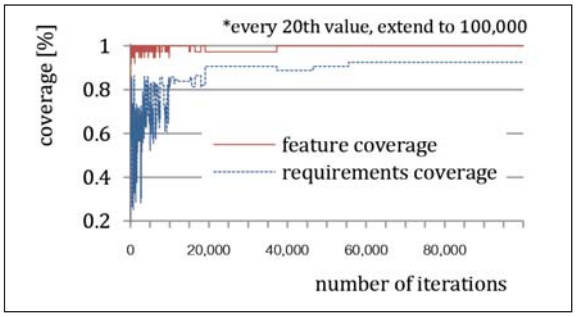

Figure 3: Simulated Annealing: Coverages at each iteration step for Case Study 1.

by applying regression strategies to reduce the overall number of test cases for the subset, such that test cases shared by variants from the subset may not have to be executed for all variants [11]. Work on this issue is still ongoing.

4 Evaluation

4.1 *Case studies*

Both methods outlined in section 3 were applied to two real world case studies at Daimler. The first case study deals with a product line from the area of thermal comfort. It has 840 requirements and 37 features, resulting in a total of approx. 10^6 variants. The second case study focuses on a product line for car battery charging systems. It has 563 requirements and 20 features, resulting in a total of approx. 10^3 variants.

Additionally, the most similar already existing method from section 2, the pairwise method, was also applied to the case studies and used as a benchmark.

4.2 *Results*

The Greedy Algorithm creates very small sets of variants in very little time. Requirements and feature coverage grow fast in the beginning (see Figure 2), which is due to the greedy approach.

In comparison, the Simulated Annealing takes longer and does not guarantee sets having 100% requirements and feature coverage. Experiments with different set sizes showed that Simulated Annealing never found qualifying sets of variants smaller than the ones found by the Greedy Algorithm. Figure 3 shows an example run for a set size equal to the result of the Greedy Algorithm.

Table 1 summarizes the results for the three methods for case study 1, while table 2 does the same for case study 2. Both case studies show the large differences in run time between the Greedy Algorithm and Simulated Annealing. The pairwise method is faster still because it does not consider requirements, only features. It always yields much larger sets of variants because it additionally demands that all valid pairs of features are present at least once.

Table 1: Results for Case Study 1

	Greedy Algorithm	Simulated Annealing	Pairwise
#variants (avg.)	8	10.6	32
run time (avg. sec)	30.8	721.2	1
req. cov. (avg. %)	100	97.33	100
feat. cov. (avg. %)	100	100	100

Table 2: Results for Case Study 2

	Greedy Algorithm	Simulated Annealing	Pairwise
#variants (avg.)	5	5	19
run time (avg. sec)	10.3	344.4	1
req. cov. (avg. %)	100	100	100
feat. cov. (avg. %)	100	100	100

5 Conclusions

In this paper, two new methods to selecting variants from a product line as test representatives for all variants were proposed. Both methods have the objective to reach full requirements coverage and full feature coverage.

The first method is based on the ideas of the Greedy Algorithm, while the second method adapts Simulated Annealing to this task. The results of both methods are compared to an already existing method based on pairwise feature coverage.

Results for two real world case studies showed that the Greedy Algorithm is superior to Simulated Annealing in terms of quality of results and run time. Even though Simulated Annealing is capable of leaving local optima in the solution space, which the Greedy Algorithm is not, Simulated Annealing never found a better solution than the Greedy Algorithm.

In comparison to the pairwise method, the number of variants selected by the new methods is significantly smaller. This is a large benefit for practical work, as testing a single variant takes significant time and cost. The fewer variants to be tested the better. Nevertheless, the full requirements and feature coverage give sufficient confidence that the test results for the variants selected are representative for all other variants of the product line.

The major advantage of the pairwise method is that it tests each interaction of two features at least once, which the new methods in this paper cannot guarantee. Therefore, for testing product lines with very high demands on functional safety, it may be useful to test additional variants selected by the pairwise method, but only after testing the variants selected by the proposed methods first. Additionally, the proposed methods could be extended to additionally optimize feature pair coverage, thus fusing the proposed approach and the pairwise method.

References

[1] K. Scheidemann, "Verifying families of system configurations," Doctoral Thesis, Technical University Munich, 2007.

[2] G. Böckle, P. Knauber, K. Pohl, and K. Schmid, *Software Product Line Engineering: Foundations, Principles and Techniques*, Springer-Verlag, 2005.

[3] M. Große-Rhode, P. Manhart, R. Mauersberger, S. Schröck, M. Schulze, and T. Weyer, "Demands of the leading industrial sectors on variability management and reuse and the resulting questions," *Software Engineering Workshop (ENVISION2020)*, 2013, pp. 251-260. (in German)

[4] E. Boutkova, "Experience with Variability Management in Requirements Specifications," in *15th International Software Product Line Conference (SPLC)*, 2011, pp. 303-312.

[5] I. do Carmo Machado, J. McGregor, and E. Santana de Almeida, "Strategies for testing products in software product lines," *SIGSOFT Software Engineering Notes*, vol. 37, pp. 1-8, 2012.

[6] S. Oster, "Feature Model-based Software Product Line Testing," Doctoral Thesis, Technical University Darmstadt, 2012.

[7] O. Manicke, "Variability management for mastering complexity in the development process of mechatronical vehicle functions," Doctoral Thesis, Technical University, Dresden, 2012. (in German)

[8] V. V. Vazirani, *Approximation Algorithms*, Springer-Verlag, 2001.

[9] A. Cmyrev and R. Reissing, "Optimized Variant and Requirements Coverage in Testing," *Informatik 2013, 11th Workshop Automotive Software Engineering*, Koblenz, pp. 2417-2429, 2013. (in German)

[10] F. W. Glover and G. A. Kochenberger, *Handbook of Metaheuristics*, Springer-Verlag, 2003.

[11] M. Lochau, I. Schaefer, J. Kamischke, and S. Lity, "Incremental Model-Based Testing of Delta-oriented Software Product Lines," in *6th International Conference on Tests and Proofs*, 2012, pp. 67-82.