# A Preliminary Empirical Assessment of Similarity for Combinatorial Interaction Testing of Software Product Lines

Stefan Fischer*, Roberto E. Lopez-Herrejon*, Rudolf Ramler†, Alexander Egyed*
* Johannes Kepler University Linz, Austria
† Software Competence Center Hagenberg, Austria
{stefan.fischer, roberto.lopez, alexander.egyed}@jku.at,
rudolf.ramler@scch.at

## ABSTRACT

Extensive work on Search-Based Software Testing for Software Product Lines has been published in the last few years. Salient among them is the use of similarity as a surrogate metric for t-wise coverage whenever higher strengths are needed or whenever the size of the test suites is infeasible because of technological or budget limitations. Though promising, this metric has not been assessed with real fault data. In this paper, we address this limitation by using Drupal, a widely used open source web content management system, as an industry-strength case study for which both variability information and fault data have been recently made available. Our preliminary assessment corroborates some of the previous findings but also raises issues on some assumptions and claims made. We hope our work encourages further empirical evaluations of Combinatorial Interaction Testing approaches for Software Product Lines.

## 1. INTRODUCTION

*Software Product Lines (SPLs)* are families of related systems whose members are distinguished by the set of features they provide [3], and their practices have shown significant technological and economic benefits [18]. *Variability* is the capacity of software artifacts to vary and its effective management and realization lies at the core of successful SPL development [20]. However, variability makes SPL testing challenging because the number of products in SPLs is typically large, and therefore it is infeasible to test every single product individually.

The last few years have seen an increasing interest in applying Search-Based techniques for *Combinatorial Interaction Testing (CIT)* of SPLs [16, 15, 14, 11]. The common thread among these approaches is to compute test suites for a subset of products in a SPL, covering feature interactions based on a *variability model* which succinctly and formally describes all the valid feature combinations of the products in a SPL. However, efficiently computing higher strength t-wise coverage (i.e. $t \geq 3$) or dealing with large SPLs (i.e., thousands of features) remains an open challenge.

Recent work by Henard et al. proposes an approach that uses a similarity metric as a surrogate metric for t-wise coverage when higher strength cannot be effectively computed or when there are limitations on the number of products that can be practically tested [13]. It is based on the intuition that the more dissimilar the products in a test suite are (i.e. the more differences amongst products in their selected and not-selected features), the higher chances of covering more t-wise feature combinations. Despite first promising results, this approach has - to the best of our knowledge - not been empirically assessed in an industry-strength study that has available both, a variability model and corresponding fault data. In this paper, we report such an assessment using Drupal [5], a widely used open source web content management system, whose variability model and fault data have recently been published by Sánchez et al. [19]. Our focus is on analyzing how faults distribute among features, how effective similarity is to detect actual faults, and how similarity compares to classical t-wise coverage. Our results corroborate some of the previous findings but also raise issues on some of the underlying assumptions and claims made.



*SBST16, May 16-17, 2016, Austin, TX, USA*



DOI: http://dx.doi.org/10.1145/2897010.2897011

## 2. EMPIRICAL ASSESSMENT OVERVIEW

Drupal is a highly modular open source web content management framework developed in PHP that has a large community of users and developers [5]. The development of Drupal uses a Git repository, an issue tracking system, and an extensive documentation which were exploited by Sánchez et al. to mine, among other data, a variability model that describes a set of valid feature combinations and a list of associated faults [19]. Drupal includes a large set of modules, more than 30,000. Each of these modules is considered a feature. In their work, Sánchez et al. focused on a small part of the Drupal functionality consisting of 48 features, for which they derived a variability model and identified 3392 faults. Table 1 summarizes the characteristics of the variability model, the number of faults they found, and the number of t-wise sets computed. The number of constraints corresponds to the number of CNF clauses obtained for the variability model; details on mapping variability models to propositional logic can be found in [4].

Our paper addresses the following three questions:

**RQ1. How are the faults distributed among features?** Rationale: For the analysis of fault detection, one

of the assumptions that Henard et al. make is that all the t-wise interactions have the same probability to trigger a fault. We are interested in analyzing whether this assumption holds true in case of Drupal.

**RQ2. What is the fault detection capability of the similarity heuristic when using Drupal's real fault data?** Rationale: Our goal is to assess how effective Henard et al.'s approach is to generate configurations that contain the feature interactions which were identified as faulty in Drupal.

**RQ3. What is the actual t-wise coverage obtained by the similarity heuristic?** Rationale: The driving goal of Henard et al.'s approach is to *mimic t-wise product configurations generation partially but efficiently while achieving decent coverage* [13]. In this regard, we are interested in comparing the coverage achieved by their approach with those implemented by an alternative tool. For this purpose, we selected the tool CASA which is capable of computing arrays of higher strength using simulated annealing [9].

| Variability Model Summary | | |
|---|---|---|
| Num Features | 48 | |
| Num Products | 2.09083392E9 | |
| Num Constraints | 77 | |
| **t value** | **t-sets** | **faults** |
| t=1 | 88 | 3232 |
| t=2 | 3,751 | 128 |
| t=3 | 103,267 | 29 |
| t=4 | 2,065,754 | 3 |
| t=5 | 32,025,724 | — |
| t=6 | 400,771,676 | — |

Table 1: Drupal Case Study Overview

# 3. ANALYSIS

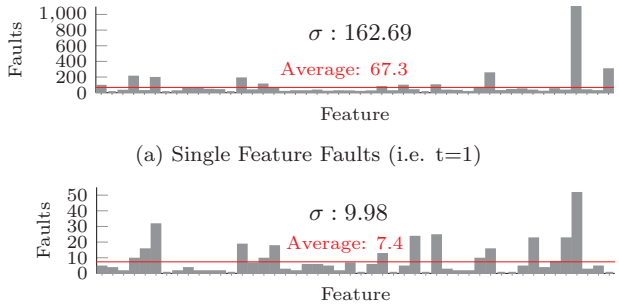
In this section we present and analyze the results obtained from running the two different CIT approaches and discuss their implications. As mentioned above, we used the similarity based CIT approach implemented by Henard et al.[1], and CASA to compute t-wise covering arrays up to t=4 [9]. All the experiments where performed on a system with an Intel Core i7-3610QM@2.30GHz, 16GB of memory, and a 64Bit environment.

## 3.1 RQ1: Distribution of Faults

Figure 1 shows for each feature the number of faults it is in, the average number of faults per feature, and the standard deviation $\sigma$. We distinguish between single feature faults (i.e. t=1, Figure 1a) and interaction faults (i.e. t=2..4, Figure 1b) in order to analyze if the distribution of faults changes when just looking into one of the two kinds of faults. Moreover the majority of faults are within single features, which means the interaction faults would not have much impact if analyzed together with the single feature faults.

In Figure 1 we can observe that the faults are not evenly distributed over the features, neither for single feature faults, nor for interaction faults. In both cases the standard deviation $\sigma$ is even greater than the average. The strongly varying

[1]http://www.research.henard.net/SPL/TSE_2014/




(a) Single Feature Faults (i.e. t=1)

(b) Interaction Faults (i.e. t=2..4)

Figure 1: Distribution of Faults

number of faults per feature indicates that the features as well as their t-wise interactions have different probabilities to trigger a fault.

Furthermore, we were also interested in the probabilities of a fault being present in a particular product. We calculated the number of products in Drupal that contain faulty t-wise interactions. We found that single feature faults appear on average in 70.5% of the products. Faults were two features interact (i.e., t=2) are on average in 55.1% of the products and faults with t= 3 and t=4 are in 48.3% and 16.5% of the products respectively. This shows that there are higher probabilities of including faults with fewer interacting features, which is not surprising since it is expected that there exist fewer products with interactions of many features.

## 3.2 RQ2: Fault Detection Capability

To assess the fault detection capability of Henard's approach we measured how many of the faults identified in Drupal are actually found (covered) by the generated sample of products. For comparison we calculate covering arrays using CASA, which stops once all the t-wise interactions are covered, hence all t-wise faults are covered. Table 2 shows the CASA run time and the size of the resulting covering arrays. We used t=1,2,3,4 since Drupal does not have faults of any higher degrees.

| $t-value$ | time | $size$ |
|---|---|---|
| 1 | 14 sec | 3 |
| 2 | 2 sec | 11 |
| 3 | 9.5 min | 38 |
| 4 | 42 hours | 126 |

Table 2: $t$-wise Covering Arrays with CASA, time for execution and size

Next we configured Henard's tool to receive as input the number of products (i.e. size of the covering array) that we obtained from CASA and performed ten runs with a fixed time limit for each run. Subsequently we calculated the number of faults that were missed by the generated products. The results are shown in Table 3. It lists the number of missed faults over all ten runs for each t between 1 and 4 in the worst (w), the average (a), and the best (b) case. In the first row of Table 3 one can see that for only three products there are some faults missed for each t. Even some single feature faults are missed here, which would have been covered in a CASA covering array. However in the next

| m | time [min] | missed faults for t | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | t=1 | | | t=2 | | | t=3 | | | t=4 | | |
| | | w | a | b | w | a | b | w | a | b | w | a | b |
| 3 | 1 | 2 | 0.4 | 0 | 20 | 12.7 | 3 | 10 | 5.4 | 1 | 3 | 1.7 | 0 |
| 11 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0.3 | 0 | 1 | 0.1 | 0 |
| 38 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 126 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 3: Faults Missed in 10 runs, for different numbers of Products $m$, w=worst, a=average, b=best run

row, with eleven products, we can observe that all faults for t=1 and t=2 are covered. Only some faults for t=3 and t=4 were missed, for which there is no guarantee that these faults would have been covered in the 2-wise CASA covering array either. Moreover, as discussed in Subsection 3.1, the probabilities of discovering faults are getting smaller the more features are needed to trigger it. Finally, the next rows show that after generating 38 and more products, all the faults are covered. This indicates that Henard's similarity approach is useful to detect interaction faults in Drupal.

### 3.3 RQ3: T-wise coverage

To answer the last research question we measured the actual coverage obtained by Henard's arrays. This tells us if similarity is a good surrogate metric for coverage in case of Drupal. We used the same runs of Henard's tool as for answering RQ2 to be able to compare the results with those of CASA for specific t-wise coverage. Note the time limits $t$ were chosen according to settings from Henard's previous work. As to be expected the coverage gets worse for higher t, as can be seen in Figure 2. Moreover for arrays sizes $m$ that were needed in a CASA covering arrays to cover all t-wise interactions, Henard's tool still missed a small percentage of them. For instance, for an array size of eleven, which was the size of the 2-wise covering array calculated with CASA, the coverage with Henard's tool was at 99%. Nonetheless this was to be expected, since Henard's approach uses similarity instead of coverage to validate the results, which will only allow an approximation at best. Nevertheless the results indicate that similarity based CIT approaches can yield a t-wise coverage comparable with CASA results.

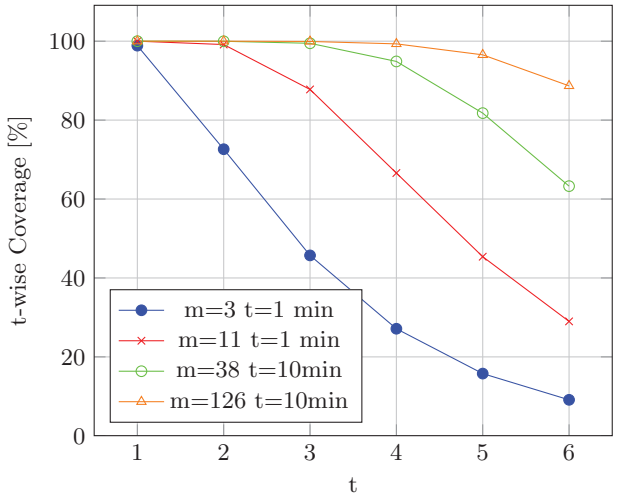



Figure 2: t-wise Coverage with respect to the number of products and time

## 4. RELATED WORK

Testing of SPLs is a topic with a large body of work as attested by two contemporary systematic mapping studies [7, 6]. These studies provide a wide overview of the area and focus on categorizing SPL testing approaches along criteria within the realm of SPL such as handling of variability and variant binding times, as well as other aspects like test organization and process.

The last few years have seen an increasing research in exploiting Search-Based techniques throughout the entire development cycle of SPLs. The survey of Harman et al. [11], and the systematic mapping study of Lopez-Herrejon et al. [16] document this increasing interest. Furthermore, both studies identified testing as the main SPL development activity where Search-Based techniques have been used.

More recently, we performed a systemic mapping study to further analyze the work on CITs for SPLs [15]. We found another approach proposed by Al-Hajjaji et al. that also uses a similarity metric[2], but in contrast with Henard et al.'s approach it is based on Hamming distance and it is applied for prioritization of covering arrays once they have been computed. In addition, this study confirmed among other things that most of the existing works focus on pairwise coverage, aim at optimizing a single objective function (e.g. minimizing test suite size or maximizing coverage), consider only information from the variability model to compute the covering arrays, and use mainly evolutionary and greedy algorithms. The latter finding goes in line with the work by Feldt and Poulding who point out that research on SBST covers only a limited number of techniques and algorithms [8], and they argue that SBST researchers should consider other approaches, more combinations and hybrid solutions. In the case of SBST of SPLs, only in a few instances (e.g. see [21]) other techniques such as swarm optimization have been explored.

Furthermore, the survey and mapping studies on Search-Based techniques applied to SPLs also point out some open research avenues such as taking a multi-objective optimization view or considering non-functional properties, both issues also raised as research avenues by the work of Harman et al. for standard (i.e. non SPL) SBST [12].

## 5. CONCLUSIONS AND FUTURE WORK

In this paper we presented our preliminary findings from assessing a similarity based CIT approach by applying it to the Drupal case study, an SPL with documented faults and a variability model. Even though we discovered that the faults in Drupal are not evenly distributed over all t-wise interactions and therefore disproved the assumption that all t-wise interactions have the same probability of triggering a fault in the case of Drupal, we still found that the similar-

ity based CIT approach produces competitive results when compared to t-wise coverage. These results seem to indicate that similarity is in fact a suitable surrogate metric for t-wise coverage, but further analysis including more diverse and larger SPLs and statistical analysis, and performing distinctive runs of the tools is needed to provide a more empirical footing to our work.

Our work can help to identify several avenues for further research. First and foremost is gathering more empirical data of other case studies that provide variability as well as fault data. We are currently extending the work by Abal et al. that identified bugs in Linux [1], by considering the evolution in both the variability model and the identified bugs. Further empirical studies will definitively help to better assess Hernard et al.'s approach and others with actual real fault data, a common practice in CIT for single systems.

Petke et al. argue that constraints are the future of CIT [17]. This is certainly the case for SPLs where constraints define the valid combinations of features, hence reducing the search space. However, from our point of view, the current work on CIT for SPLs has not sufficiently explored constraints that stem from other sources such as code artifacts (e.g. method dependencies) or non-functional properties (e.g. feature LOC size) that could help to better constrain the search spaces for SBST of SPLs. For instance Sánchez et al. discovered that for Drupal there is a correlation between the number of faults reported for a feature and the size of that feature in lines of code [19]. We believe it would be interesting to perform such analysis also on other SPLs, to check if such metrics can be a suitable to improving the sampling of products for testing.

## 6. ACKNOWLEDGMENTS

We thank Christopher Henard for this help clarifying questions about their work and tool support. We also thank Ana Sánchez and her group for providing the data of the Drupal case study. Stefan Fischer is a recipient of a DOC Fellowship of the Austrian Academy of Sciences at the Institute for Software Systems Engineering. This research was partially funded by the Austrian Science Fund (FWF) projects P25289-N15, P25513-N15. The research reported in this paper has been partly supported by the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry of Science, Research and Economy, and the Province of Upper Austria in the frame of the COMET center SCCH.

## 7. REFERENCES

[1] I. Abal, C. Brabrand, and A. Wasowski. 42 variability bugs in the linux kernel: a qualitative analysis. In I. Crnkovic, M. Chechik, and P. Grünbacher, editors, *ASE*, pages 421–432. ACM, 2014.

[2] M. Al-Hajjaji, T. Thüm, J. Meinicke, M. Lochau, and G. Saake. Similarity-based prioritization in software product-line testing. In *SPLC*. ACM, 2014.

[3] D. S. Batory, J. N. Sarvela, and A. Rauschmayer. Scaling step-wise refinement. *IEEE Trans. Software Eng.*, 30(6):355–371, 2004.

[4] D. Benavides, S. Segura, and A. R. Cortés. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, 35(6):615–636, 2010.

[5] D. Buytaert. Drupal Framework. http://www.drupal.org, accessed in October 2015.

[6] I. do Carmo Machado, J. D. McGregor, Y. C. Cavalcanti, and E. S. de Almeida. On strategies for testing software product lines: A systematic literature review. *Information and Software Technology*, 56(10):1183 – 1199, 2014.

[7] E. Engström and P. Runeson. Software product line testing - a systematic mapping study. *Inform. & Software Tech.*, 53(1):2–13, 2011.

[8] R. Feldt and S. M. Poulding. Broadening the search in search-based software testing: It need not be evolutionary. In Gay and Antoniol [10], pages 1–7.

[9] B. Garvin, M. Cohen, and M. Dwyer. Evaluating improvements to a meta-heuristic search for constrained interaction testing. *Empirical Software Engineering*, 16(1):61–102, 2011.

[10] G. Gay and G. Antoniol, editors. *8th IEEE/ACM International Workshop on Search-Based Software Testing, SBST 2015, Florence, Italy, May 18-19, 2015*. IEEE, 2015.

[11] M. Harman, Y. Jia, J. Krinke, W. B. Langdon, J. Petke, and Y. Zhang. Search based software engineering for software product line engineering: a survey and directions for future work. In *SPLC*, 2014.

[12] M. Harman, Y. Jia, and Y. Zhang. Achievements, open problems and challenges for search based software testing. In *ICST*, pages 1–12. IEEE, 2015.

[13] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. L. Traon. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. *IEEE Trans. Software Eng.*, 40(7):650–670, 2014.

[14] R. E. Lopez-Herrejon, J. Ferrer, F. Chicano, A. Egyed, and E. Alba. *Computational Intelligence and Quantitative Software Engineering*, chapter Evolutionary Computation for Software Product Line Testing: An Overview and Open Challenges. Springer, 2015. Accepted for publication.

[15] R. E. Lopez-Herrejon, S. Fischer, R. Ramler, and A. Egyed. A first systematic mapping study on combinatorial interaction testing for software product lines. In *ICST Workshops*, pages 1–10, 2015.

[16] R. E. Lopez-Herrejon, L. Linsbauer, and A. Egyed. A systematic mapping study of search-based software engineering for software product lines. *Journal of Information and Software Technology*, 2015.

[17] J. Petke. Constraints: The future of combinatorial interaction testing. In Gay and Antoniol [10].

[18] K. Pohl, G. Bockle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.

[19] A. B. Sánchez, S. Segura, J. A. Parejo, and A. Ruiz-Cortés. Variability testing in the wild: The drupal case study. *Software and Systems Modeling Journal*, pages 1–22, Apr 2015.

[20] M. Svahnberg, J. van Gurp, and J. Bosch. A taxonomy of variability realization techniques. *Softw., Pract. Exper.*, 35(8):705–754, 2005.

[21] S. Wang, S. Ali, and A. Gotlieb. Cost-effective test suite minimization in product lines using search techniques. *Journal of Systems and Software*, 103:370–391, 2015.