**Evaluating Testing Techniques in Highly-Configurable Systems: The Drupal Dataset**

*Ana B. Sánchez, Sergio Segura and Antonio Ruiz Cortés*

Department of Computer Languages and Systems, University of Seville, Seville, Spain

anabsanchez@us.es, sergiosegura@us.es, aruiz@us.es

**Abstract**

**Context***:* Software applications exposing a high ability to be extended, changed or configured are usually referred to as Highly-Configurable Systems (HCSs). Testing techniques for *HCSs* aim at finding effective but manageable test suites that lead to the early detection of faults. Evaluating the effectiveness of these techniques in realistic environments is a must, but also a challenge due to the lack of HCSs with available code, configuration models and fault reports. **Aim**: In this chapter, we present the Drupal dataset, a collection of real-world data collected from the popular open-source Drupal framework. This dataset allows the assessment of variability testing techniques with real data of an HCS. **Method**: We collected extensive non-functional data from the Drupal Git repository and the Drupal website, including code changes in different versions of Drupal modules (e.g., 557 commits in Drupal v7.22) and number of tests and assertions in the modules (e.g., 352 and 24,152, respectively). The faults found in different versions of Drupal modules were also gathered from the Drupal bug tracking system (e.g., 3,392 faults in Drupal v7.23). Additionally, we provided the Drupal feature model as a representation of the framework configurability, with more than 2000 millions of different Drupal configurations; one of the largest attributed feature model published so far. With 125 citations since its publication, the Drupal dataset has become a helpful tool to researchers and

practitioners to conduct more realistic experiments and evaluations of HCSs.

**Keywords**

Highly-configurable systems, variability, testing, dataset, feature model, Drupal

## Background & Summary

*Highly-Configurable Systems (HCS)* determines the ability of software applications to be extended, customized or configured [1]. Operating systems such as Linux [2] or development tools such as Eclipse [3] have been reported as examples of HCSs. Another prominent example of HCS is Software Product Lines (SPL) [4]. SPL engineering focuses on the development of families of related products through the systematic management of variability. For that purpose, *feature models* are typically used as the de facto standard for configurability modelling in terms of functional features and constraints among them [5]. The number of configurations in these models is potentially huge. This makes testing HCSs a challenge task. To address this problem, researchers have proposed numerous techniques to reduce the cost of testing in the presence of variability [1,3,6,7,8,9]. To evaluate these techniques, unrealistic experiments are usually carried out by researchers that employ synthetic feature models and data that introduce threats to validity and question their conclusions. This is due to the lack of real-world data available about HCSs that share code, test cases, detailed fault report or even a detailed documentation that enables the reproducibility of experiments in realistic environments [2].

In order to search for real-world HCSs with available code we followed the steps of previous authors and looked into the open source community. Particularly, we found the popular open-source Drupal framework, a highly modular web content management written in PHP [10,11].

Drupal has more than 30,000 modules that can be composed to form valid configurations of the system. Drupal provides detailed fault reports including fault description, fault severity, type, status and so on. The high number of the Drupal community members together with its extensive documentation have also been strengths to choose this framework, currently maintained and developed by a community of more than 630,000 users and developers. Drupal can be used to build a variety of web sites including internet portals, e-commerce applications and online newspapers. Drupal is composed of a set of modules. A module is a collection of functions that provide certain functionality to the system. According to the Drupal documentation, each module that is installed and enabled adds a new feature to the framework [11]. Thus, we propose modelling Drupal modules as features of the feature model.

In this chapter, we present the Drupal dataset, a publicly available resource of valuable testing data about the Drupal framework and its modules. In particular, we provide the following information:

1. *The Drupal feature model*. We model some of the main Drupal modules to features and represent the framework configurability using a feature model. The resulting model has 48 features, 21 cross-tree constraints and represents 2.09*E*9 different Drupal configurations.
2. *Non-functional Drupal data*. We report on extensive non-functional data extracted from the Drupal Git repository. For each feature under study, we report its size, number of changes (during two years), cyclomatic complexity, number of test cases, number of test assertions, number of developers and number of reported installations. The non-functional data are modelled as feature attributes in the feature model.

3. *History of Drupal faults*. We present the number of faults reported on the Drupal features under study during a period of two years, extracted from the Drupal issue tracking system. Faults are classified according to their severity and the feature(s) that trigger it. Among other results, we identified 3392 faults in Drupal v7.23, 160 of them caused by the interaction of up to four different features. We replicated the study of faults in two consecutive Drupal versions, v7.22 and v7.23, to enable fault history test validations.

As far as we know, the Drupal dataset has been used by numerous research articles to evaluate their HCS testing techniques. Proof of this are the 125 citations received since its first research publications [12,13,14]. Among others researches, Hierons et al. proposed a new testing technique to solve the many-objective optimisation problem that was evaluated with the Drupal dataset [15]. Previously, the same authors presented a proposal for obtaining the optimal selection of products from feature models using many-objective evolutionary optimization. The feasibility of this approach was assessed with data that included the Drupal dataset [16]. Fischer et al. performed an empirical assessment of similarity for testing software product lines and the Drupal dataset was key in their evaluation [9].

**Dataset Specification**

| | |
|---|---|
| **Subject** | Testing of highly-configurable systems |
| **Specific subject area** | Automated test case selection, prioritization and minimization of highly-configurable systems. |
| **Type of data** | **Table** |

| | |
|---|---|
| | **Model**<br>**Figure** |
| **How data were acquired** | Repository and bug tracking mining and literature review<br>Software, console commands and manual data review |
| **Data format** | Link to the Drupal DataSet and experiments:<br>https://github.com/belene/DrupalDataset<br>Contains:<br>Drupal feature model in SXFM format (XML format)<br>Drupal feature model in FaMa format (XML format)<br>Drupal feature data (CSV format)<br>Drupal feature faults (CSV format)<br><br>Analyzed<br>Filtered |
| **Parameters for data collection** | We designed the Drupal feature module using 48 modules and 21 dependencies identified among the modules. To analyse the effectiveness of non-functional data as bug predictors, we collected the commits made in modules and the faults recorded in two different versions of Drupal and in the period of two years, obtaining a total of 557 changes and 3,301 faults for Drupal v7.22. |
| **Description of data** | We followed a systematic approach and mapped each Drupal |

| | |
|---|---|
| **collection** | module to a feature. We also used the dependencies defined in the information file of each Drupal module to model cross-tree constraints. We filtered the faults by feature name, framework version and dates. Then, the search was refined to eliminate the faults not accepted by the Drupal community. Next, we manually checked the bug reports of each candidate integration fault and discarded those not correctly identified. To obtain the number of changes made in each feature, we tracked the commits to the Drupal Git repository by using console commands. The rest of the data was obtained through the review of the official Drupal documentation and websites. |
| **Data source location** | Institution: University of Seville<br>City: Seville<br>Country: Spain |
| **Data accessibility** | The data are publicly available in a repository. But part of the data and their explanation are included in the article.<br>Repository name: The Drupal Dataset<br>Direct URL to data: https://github.com/belene/DrupalDataset |
| **Related research article** | Sánchez, Ana B., Segura, Sergio, Parejo, José A., Ruiz-Cortés, Antonio. Variability testing in the wild: The drupal case study. Software & Systems Modeling. 1 (2017) 173-194. https://doi.org/10.1007/s10270-015-0459-z |

| | |
|---|---|
| | Ana B. Sánchez, Sergio Segura, and Antonio Ruiz Cortés. *The Drupal Framework: A Case Study to Evaluate Variability Testing Techniques*. Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems. https://doi.org/10.1145/2556624.2556638 |

**Value of the Data**

- Drupal dataset enables the evaluation of testing techniques for highly-configurable systems with real-world data from the community open-source.
- Both, researchers and practitioners can benefit from these data. This dataset provides variability researchers and practitioners with helpful information about the distribution of faults and test cases in a real highly-configurable system. Also, it is a valuable asset to evaluate HCS testing techniques in realistic settings rather than using random variability models and simulated faults.
- The data collected in the Drupal dataset can be used as good indicators of the fault propensity of a software application by using the history of faults and changes in different versions of Drupal.
- The Drupal dataset can be useful in any analysis work of feature models.

**Data Description**

The Drupal dataset is publicly available in a Github repository:

https://github.com/belene/DrupalDataset

- *DRUPALv4SXFM.xml:* Drupal feature model in SXFM format. The model comprises 48 features and 21 constraints. It can be found in the folder *FMs* of the *DrupalDataset* repository.
- *DRUPALv4FAMA.xml:* Drupal feature model in FaMa format. The model comprises 48 features and 21 constraints. It can be found in the folder *FMs* of the *DrupalDataset* repository.
- *DrupalFeaturesData.csv*: A CSV File containing the Drupal modules considered in the dataset and, for each module, the size, the code cyclomatic complexity, the number of test cases and assertions, the number of reported installations, the number of commits made in Drupal versions v7.22 and v7.23, and the number of faults recorded in Drupal v7.22 and v7.23.
- *DrupalFeatureFaults.csv:* A CSV file containing for each Drupal module the number of collected faults classified by type (single or integration faults), severity (minor, normal, major and critical) and by Drupal version (v7.22 and v7.23).

**Experimental Design, Materials, and Methods**

*Drupal feature model*. According to the Drupal documentation, each module that is installed and enabled in the system adds a new feature to the framework [11]. Thus, we followed a systematic approach and proposed modelling Drupal modules as features of the feature model. Also, when a module is installed, new subfeatures can be enabled adding extra functionality to the module. These features are considered as children features of the module that contains them. The Drupal core modules that must be always enabled are represented as mandatory relations in the feature model. On the other side, all the modules that can be optionally installed and enabled in the

system are modelled as optional features in the model. In addition to this, Drupal modules can have dependencies with other modules, i.e. modules that must be installed and enabled for another module to work properly. These dependencies are modelled as cross-tree constraints in the form of requires in the feature model.

*Non-functional data*. Drupal dataset also reports a number of non-functional attributes of the features selected for the Drupal feature models, namely:

- *Feature size*. This provides a rough idea of the complexity of each feature and its fault propensity. The size of a feature was calculated in terms of the number of lines of code (LoC).
- *Cyclomatic Complexity (CC)*: This metric reflects the total number of independent logic paths used in a program and provides a quantitative measure of its complexity. We used the open-source tool *phploc* to compute the CC of the source code associated with each Drupal feature. Roughly speaking, the tool calculates the number of control flow statements (e.g. “if”, “while”) per lines of code.
- *Number of tests*. We provide the total number of test cases and test assertions of each Drupal feature obtained from the output of the *SimpleTest* module.
- *Number of reported installations.* This depicts the number of times that a Drupal feature has been installed as reported by Drupal users. This data was extracted from the Drupal website [10] and could be used as an indicator of the popularity or impact of a feature.
- *Number of developers*. We collected the number of developers involved in the development of each Drupal feature. This could give us information about the scale and relevance of the feature as well as its propensity to faults related to the number of people

working on it. This information was obtained from the website of each Drupal module as the number of committers involved [10].

- *Number of changes.* Changes in the code are likely to introduce faults. Thus, the number of changes in a feature may be a good indicator of its error proneness and could help us to predict faults in the future. To obtain the number of changes made in each feature, we tracked the commits to the Drupal Git repository.

*Faults in Drupal*. The Drupal dataset collects the number of faults reported in the Drupal features. The information was obtained from the issue tracking systems of Drupal and related modules. We used the web-based search tool of the issue systems to filter the bug reports by severity, status, date, feature name and Drupal version. The search was narrowed by collecting the bugs reported in a period of two years and in two consecutive Drupal versions, v7.22 and v7.23, to achieve a better understanding of the evolution of a real system and to enable test validations based on fault history. Then, the search was refined to eliminate the faults not accepted by the Drupal community, those classified as duplicated bugs, non-reproducible bugs and bugs working as designed. Additionally, we identified and classified the faults into single (those caused by a Drupal model) and integration faults (those caused by the interaction of several modules). To mitigate possible misidentification of faults, we manually checked each candidate integration fault to discard those that did not correspond.

For the sake of validation, the work was discussed with two Drupal core team members who approved the followed approach. We also may mention that the main author of the article has more than two year of experience in industry as a Drupal developer.

**Acknowledgments**

This work has been partially supported by the European Commission (FEDER), the Spanish Government under the project HORATIO (RTI2018-101204-B-C21) and the Andalusian R&D&I programs APOLO (US-1264651) and EKIPMENT (PR18-FR-2895).

**Competing Interests**

The authors declare that they have no known competing financial interests or personal relationships which have, or could be perceived to have, influenced the work reported in this article.

**References**

[1] Cohen, M. B., Dwyer, M. B., Shi, J. Constructing Interaction Test Suites for Highly-Configurable Systems in the Presence of Constraints: A Greedy Approach. Transactions on Software Engineering. 5 (2008), 633-650. https://doi.org/10.1109/TSE.2008.50.

[2] She, S., Lotufo, R., Berger, T., Wasowski, A., Czarnecki, K. The variability model of the linux kernel. International Workshop on Variability Modelling of Software-Intensive Systems. 2010.

[3] Johansen, M.F., Haugen, O., Fleurey, F., Eldegard, A.G., Syversen, T. Generating better partial covering arrays by modeling weights on sub-product lines. International Conference on Model Driven Engineering Languages and Systems. 2012. pp. 269–284. https://doi.org/10.1007/978-3-642-33666-9_18.

[4] Svahnberg, M., van Gurp, L., Bosch, J. A taxonomy of variability realization techniques: research articles. Softw. Pract. Exp. 35 (2005) 705–754.

https://dl.acm.org/doi/10.5555/1070904.1070905.

[5] Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S. Feature-oriented domain analysis (foda) feasibility study. Software Engineering Institute. 1990.

[6] Yoo, S., Harman, M. Regression testing minimisation, selection and prioritisation: a survey. Software Testing, Verification and Reliability. (2012) 67–120. https://doi.org/10.1002/stv.430.

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

[8] Sánchez, A.B., Segura, S., Ruiz-Cortés, A. A comparison of test case prioritization criteria for software product lines. IEEE International Conference on Software Testing, Verification, and Validation. 2014 pp. 41–50. https://doi.org/10.1109/ICST.2014.15.

[9] Fischer, S., Lopez-Herrejon, R. E., Ramler, R., Egyed, A. A Preliminary Empirical Assessment of Similarity for Combinatorial Iteraction Testing of Software Product Lines. IEEE/ACM 9th International Workshop on Search-Based Software Testing (SBST), 2016, pp. 15-18.

[10] Buytaert, D. Drupal framework. http://www.drupal.org. Accessed April 2020.

[11] Tomlinson, T., VanDyk, J.K. Pro Drupal 7 Development. Tomlinson, Todd (et al.). 3rd edition. 2010.

[12] Sánchez, Ana B., Segura, Sergio, Ruiz-Cortés, Antonio. The drupal framework: A case study to evaluate variability testing techniques. Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems. 2014, pp. 1-8. https://doi.org/10.1145/2556624.2556638.

[13] Sánchez, Ana B., Segura, Sergio, Parejo, José A., Ruiz-Cortés, Antonio. Variability testing

in the wild: The drupal case study. Software & Systems Modeling. 1 (2017) 173-194. https://doi.org/10.1007/s10270-015-0459-z.

[14] Parejo, José A., Sánchez, Ana B., Segura, Sergio, Ruiz-Cortés, Antonio, Lopez-Herrejon, Roberto E., Egyed, Alexander. Multi-objective test case prioritization in highly configurable systems: A case study. Journal of Systems and Software. (2016) 287-310. https://doi.org/10.1016/j.jss.2016.09.045.

[15] Hierons, Robert M., Li, Miqing, Liu, Xiaohui, Parejo, Jose Antonio, Segura, Sergio, Yao, Xin. Many-Objective Test Suite Generation for Software Product Lines. ACM Trans. Softw. Eng. Methodol. 1 (2020) 46 pages. https://doi.org/10.1145/3361146.

[16] Hierons, Robert M., Li, Miqing, Liu, Xiaohui, Segura, Sergio, and Zheng, Wei. 2016. SIP: Optimal Product Selection from Feature Models Using Many-Objective Evolutionary Optimization. ACM Trans. Softw. Eng. Methodol. 17 (2016), 39 pages. https://doi.org/10.1145/2897760.