SmarTest: A Test Case Prioritization Tool for Drupal

Ana B. Sánchez and Sergio Segura
Department of Computer Languages and Systems
University of Seville
Seville, Spain
{anabsanchez,sergiosegura}@us.es

ABSTRACT

Test case prioritization techniques aim to identify the optimal ordering of tests to accelerate the detection of faults. The importance of these techniques has been recognized in the context of Software Product Lines (SPLs), where the potentially huge number of products makes testing extremely challenging. We found that the open source Drupal framework shares most of the principles and challenges of SPL development and it can be considered a real-world example of family of products. In a previous work, we represented the Drupal configuration space as a feature model and we collected extra functional information about its features from open repositories. Part of this data proved to be a good indicator of faults propensity in Drupal features. Thus, they become valuable assets to prioritize tests in individual Drupal products. In this paper, we present SmarTest, a test prioritization tool for accelerating the detection of faults in Drupal. SmarTest has been developed as an extension of the Drupal core testing system. SmarTest supports the prioritization of tests providing faster feedback and letting testers begin correcting critical faults earlier. Different test prioritization criteria can be selected in SmarTest, such as prioritization based on the number of commits made in the code, or based on the tests that failed in last executions. A customizable dashboard with significant system information to guide the testing is also provided by SmarTest at run-time. This work represents an interesting application of SPL-inspired testing techniques to real-world software systems, which could be applicable to other open-source SPLs.

KEYWORDS

Software product lines,

Variability,

Tool,

Prioritization,

Testing

1 INTRODUCTION

A Software Product Line (SPL) is a family of related software products. Each product represents a specific combination of features of the SPL. In previous works, we found the open source Drupal framework to be a motivating example of a real highly-configurable software system [9, 11, 12]. Drupal is a highly modular web content management framework written in PHP [2, 14]. Drupal provides extensive documentation about its modules and detailed fault reports including fault description, severity, type, status, etc. Furthermore, it is maintained and developed by a community of more than 630,000 users and developers. Although Drupal is not the result of an SPL engineering approach, it shares most of the principles and challenges of SPL development. Hence, Drupal products are created by composing modules, described as "features" in the Drupal documentation, where each feature is an increment in product functionality [14]. Also, as in SPLs, Drupal modules (i.e., features) can be divided into core features (i.e., core compulsory modules) and optional features (i.e., core optional and additional modules) and can present restrictions among them.

Although software development based on SPLs provides many advantages, such as reduction of the overall software development costs, it challenges quality assurance. Testing SPLs is extremely hard to manage due to the potentially huge number of products under test [4, 8]. For instance, Drupal provides more than 30,000 modules that can be combined with restrictions leading to billions of potential different products [14]. This often makes testing all the products in an SPL infeasible. To alleviate this problem, numerous techniques have been proposed to reduce the test space to a manageable subset of products to be tested [3, 5, 7]. Other studies have presented test prioritization techniques to find the optimal order in which the products should be tested to detect faults faster [1, 6].

In previous works, we presented an industry-strength case study based on Drupal with more than 2 billion of configurations to be used as a realistic subject for further and reproducible validation of variability testing techniques [9, 11]. We mined the Drupal repositories to extract significant information from the system. Among other results, we identified 3,392 faults in single features and 160 faults triggered by the interaction of up to four features in Drupal v7.23. We also found positive correlations relating the number of bugs in Drupal features to their size, cyclomatic complexity, number of changes and fault history. Our evaluations showed that extra functional system properties are effective drivers to accelerate the detection of faults, outperforming other related prioritization criteria as test case similarity. In addition to this, we observed that the official tool to test Drupal modules, named SimpleTest[1], suffers from several drawbacks, essentially: 1) it is not a customizable

[1] www.drupal.org/simpletest

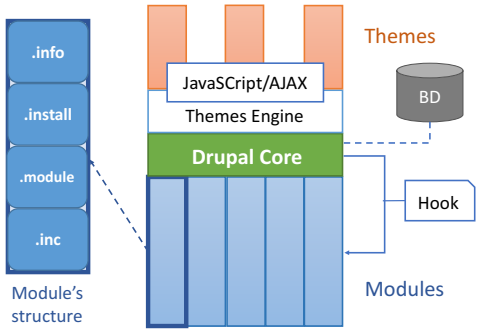


Figure 1: Drupal architecture and module structure.

and configurable tool, 2) it does not return any feedback from test executions while the tests are running, and 3) it does not support test case prioritization. This is crucial since running all the tests in Drupal can take many hours (e.g. 10 hours in a basic Drupal product with 40 modules) and it could be useful to prioritize the tests to detect as many faults as possible sooner.

Considering previous findings, we found that the information collected for testing Drupal at the family level is also helpful to test individual Drupal products, creating an interesting loop where individual Drupal-based applications are more effectively tested using the feedback obtained from the whole family of products. This idea can provide to SPL community a valuable example on how to manage the complexity of testing a real-world SPL. In this line, we have developed SmarTest, a test prioritization tool for accelerating the detection of faults in Drupal.

The paper is structured as follows. We describe an overview of Drupal and its core testing system in Section 2. We present SmarTest, its capabilities and structure within Drupal in Section 3. Finally, we summarize the conclusions and future work in Section 4.

2 SIMPLETEST IN DRUPAL

Drupal is an open source PHP framework used to build a variety of web sites including internet portals and e-commerce applications. Figure 1 shows an overview of the Drupal architecture, which is composed of the following main elements: the core, modules, hooks and themes [14]. The core includes code that allows the Drupal system to bootstrap when it receives a request, a library of common functions frequently used with Drupal, and core modules that provide basic functionality like user management, templating and testing. A Drupal product is composed of a set of modules, which can be classified into core modules and additional modules. Core modules are included by default in the basic installation of the Drupal framework. Additional modules can be optionally installed and enabled. These modules are developed by the Drupal community and shared under the same GNU Public License (GPL) as Drupal.

Drupal is built on a system of hooks, sometimes called callbacks. Hooks are how modules can interact with the core code of Drupal. Hooks occur at various points in the thread of execution, where Drupal seeks contributions from all the enabled modules. The theme layer in Drupal is responsible for creating the HTML (or JSON, XML, etc.) that the browser will receive. Drupal uses PHP Template as the primary templating engine [14].

SimpleTest is the only core module for testing in Drupal. It permits to define and run the tests of all the installed modules automatically. SimpleTest reads the tests defined in a Drupal product and shows a listing of tests grouped by categories, such as cache-based or user-based tests, as illustrated in Figure 2(a). Then, it enables the selection or deselection of the tests to be executed. Although SimpleTest is a powerful testing module, it has the following limitations:

Lack of historical data. SimpleTest presents just a list of tests to be executed (see Figure 2(a)). It does not show any information to the tester about historical data such as previous test executions or about the modules containing modifications that need to be tested.

Limited tests configuration functionality. SimpleTest only allows the selection or deselection of tests to be run. It does not provide any features to configure the following test execution, running the tests always in the same established order.

No feedback at run-time. SimpleTest does not return any feedback until all tests finish their executions. This is relevant since running all the tests in Drupal can take many hours.

3 SMARTEST

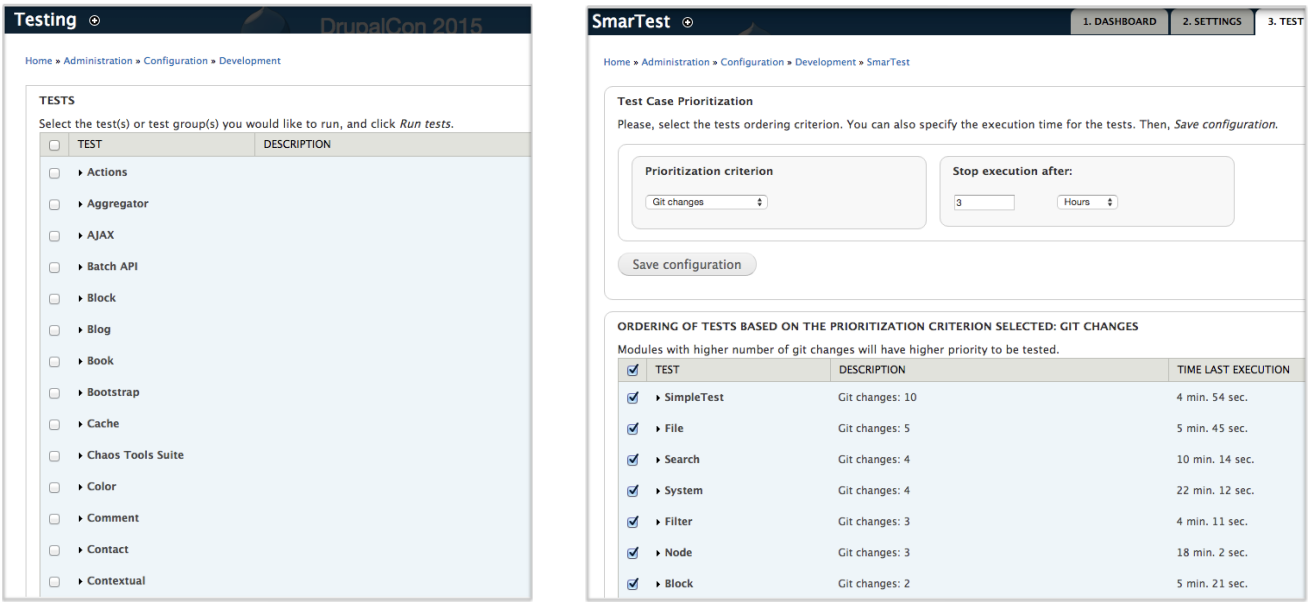
SmarTest has been developed as an extension of the Drupal core module SimpleTest. SmarTest can be considered a contributed module in Drupal within the category of additional modules. It is developed using hooks to extend and enhance the SimpleTest functionality. The tool is freely distributed under GNU GPL v2 license and can be downloaded from the official Drupal projects page[2].

SmarTest, like every Drupal module, is mapped to a directory including source files such PHP files, CSS stylesheets, JavaScript code and help documents. Each Drupal module must include a .info file and a .module file, whose names must match the name of the module. The .info file contains information about the module such as the module description, PHP and Drupal versions, dependencies with other modules, etc. The .module file is the main PHP file that contains all of the code. Also, modules can include a .install file that it is run the first time the module is enabled, and some .inc PHP files including specific functionality. The structure of a Drupal module is showed in Figure 1. Drupal modules can also include a test directory with the test cases associated with the module.

In general, SmarTest enables the automated analysis of the system under test offering useful information to guide software engineers, and it supports the automated prioritization and execution of all the tests defined in the modules of the system. More specifically, current version of SmarTest provides the following features:

Dashboard with information at run-time. SmarTest presents a dashboard that enables the display of actual information about the Drupal modules installed and enabled in the system, such as their size (in terms of lines of code), the number of tests per module that passed and failed in last tests executions or the complexity of the modules. This data has been showed to be closely related with the propensity to faults of Drupal modules [11]. Additionally, the dashboard shows the percentage of code coverage of the tests, the

[2] www.drupal.org/project/smartest



(a) SimpleTest module

(b) SmarTest module

Figure 2: User interfaces for testing in Drupal

modules with less test coverage, the modules with more failures detected in last executions, the percentage of tests passed and failed in last run or even the time taken by the last test execution. Figure 3 illustrates a screenshot of a dashboard created in SmarTest.

Customizable dashboard. SmarTest's dashboard is totally customizable and configurable through widgets functionality. This means that testers can edit, delete or add new widgets of information (e.g., a chart showing the relation between lines of code in modules and their cyclomatic complexity). Furthermore, SmarTest allows testers to select the desired format to show the information (e.g., tag clouds, column graphs or row graphs, number of modules to show, etc.). Some customized widgets are illustrated in Figure 3.

Prioritization technique selector. SmarTest supports the use of different test prioritization techniques based on real data of the Drupal product under test. In particular, we can order the test execution based on: 1) the cyclomatic complexity of the modules, 2) the number of commits made in the modules (taken from the Drupal project GitHub automatically), 3) the code covered by the tests, 4) the number of tests that failed or threw exceptions in last executions, and 5) the size of the modules. The prioritization selector is showed on the top left of the Figure 2(b).

Time-out execution button. SmarTest provides a stop button to optionally indicate the time (given in minutes or hours) in which the tests will be running. Figure 2(b) illustrates this button on the top right, specifying 3 hours of test execution.

Test case prioritization. Once we have indicated the test prioritization technique and the execution time, SmarTest displays all the tests of the enabled modules ordered by the prioritization criterion selected, showing the prioritization value and the time taken in the last run for each group of tests. Then, the system allows us to select the tests to be run in the established order. As an example, consider the Figure 2(b), where the tester selected the prioritization criterion driven by Git changes to guide the testing. Thus, the modules in Drupal with higher number of Git changes will have higher priority to be tested.

Automated testing with continuous feedback at real time. It is noteworthy that when the execution of a test finishes, SmarTest returns a detailed result report immediately. This enables the continuous feedback on the testing progress in real time, which allows software engineers to start fixing the bugs earlier. Note that running all the tests in Drupal can take many hours or even days.

SmarTest has been warmly welcomed by the Drupal Community. Evidence of this was the presentation of our project in the National Conference DrupalCampSpain [13], and the later invitation to present SmarTest in the International Conference DrupalConEurope [10], where thousands of Drupal professionals meet annually. It is worth mentioning that SmarTest counts on the assistance of some Drupal professional altruistic contributors interested in the development of a more advanced testing tool. The SmarTest module is installed and enabled in a Drupal product and it can be checked at http://www.isa.us.es/smartest/demo-instructions.html following the instructions.

4 CONCLUSIONS AND FUTURE WORK

In this paper, we have presented SmarTest, a test prioritization tool for Drupal framework. Our goal is to make our research accessible and useful to both the academic and the industrial community. Our

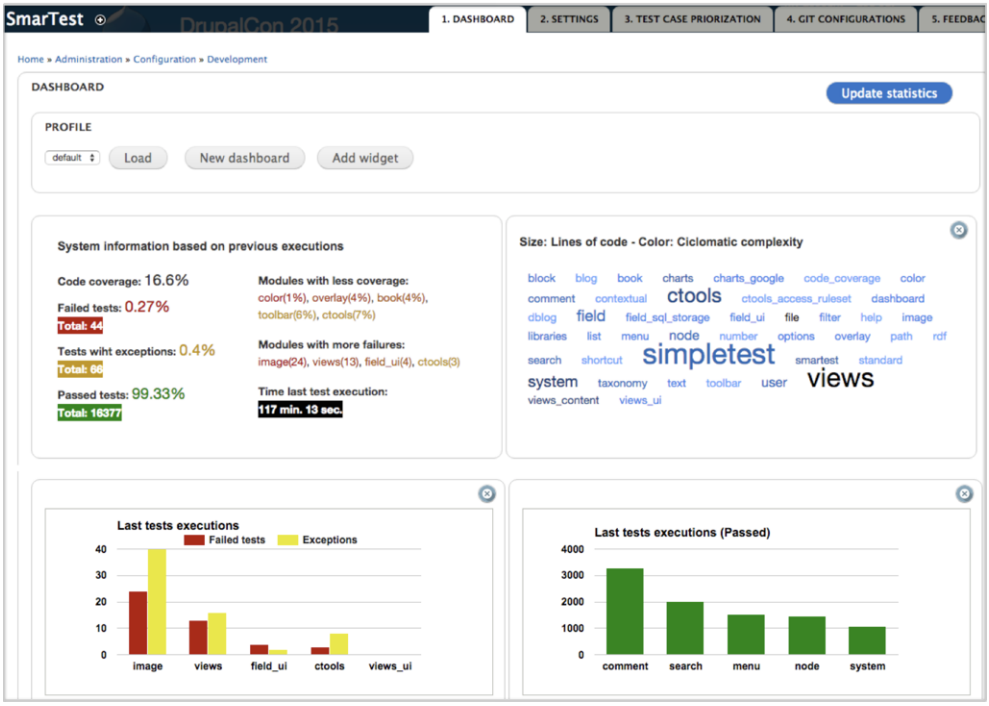


Figure 3: SmarTest's dashboard

tool is an example of how open source communities can benefit from the SPL techniques, and how SPL community can take advantage of the problems detected in open source highly-configurable systems. SmarTest presents relevant improvements with respect to Simpletest, the core testing module in Drupal, namely: 1) a customizable dashboard with valuable system information to guide the testing and 2) test case prioritization with continuous feedback at run-time enabling early bug fixes. SmarTest is distributed under the GNU GPL v2 license recommended by Drupal to allow other Drupal developers take part in its development. SmarTest has been presented in the National Conference DrupalCampSpain and the International Conference DrupalConEurope with very positive acceptance from the community. It is remarkable that SmarTest is currently being reviewed and used by some companies and Drupal professionals. We trust that our experience could be replicated in other open source SPLs, where the data extracted from the family of products can be used to select and prioritize tests on individual products in a more effective way.

We plan to work on adapting SmarTest, currently working on Drupal 7, to Drupal 8. We also plan to integrate our multi-objective prioritization techniques into SmarTest since we are confident that this could further improve the process of testing in Drupal.

ACKNOWLEDGMENTS

We thank Gabriel Hidalgo for his development work on SmarTest and the Drupal contributors for their helpful assistance and contributions. This work was partially supported by the European Commission (FEDER) and the Spanish and Andalusian R&D&I programmes (BELI TIN2015-70560-R and COPAS P12-TIC-1867).

REFERENCES

[1] M. Al-Hajjaji, T. Thum, J. Meinicke, M. Lochau, and G. Saake. 2014. Similarity-Based Prioritization in Software Product-Line Testing. In *Software Product Line Conference*. 197–206.
[2] D. Buytaert. accessed in May 2017. Drupal Framework. http://www.drupal.org. (accessed in May 2017).
[3] X. Devroey, G. Perrouin, and P. Schobbens. 2014. Abstract test case generation for behavioural testing of software product lines. In *Software Product Line Conference*, Vol. 2. ACM, 86–93.
[4] I. do Carmo Machado, J. D. McGregor, Y. Cerqueira Cavalcanti, and E. Santana de Almeida. 2014. On strategies for testing software product lines: A systematic literature review. *Information and Software Technology* 56, 10 (2014), 1183 – 1199.
[5] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. Le Traon. 2013. Multi-objective Test Generation for Software Product Lines. In *International Software Product Line Conference*. 62–71.
[6] R. E. Lopez-Herrejon, J. Ferrer, F. Chicano, E. N. Haslinger, A. Egyed, and E. Alba. 2014. A parallel evolutionary algorithm for prioritized pairwise testing of software product lines. In *Genetic and Evolutionary Computation Conference*. 1255–1262.
[7] D. Marijan, A. Gotlieb, S. Sen, and A. Hervieu. 2013. Practical Pairwise Testing for Software Product Lines. In *Software Product Line Conference*. ACM, 227–235.
[8] A. Metzger and K. Pohl. 2014. Software Product Line Engineering and Variability Management: Achievements and Challenges. In *Proceedings of the on Future of Software Engineering (FOSE 2014)*. ACM, New York, NY, USA, 70–84.
[9] J. A. Parejo, A. B. Sánchez, S. Segura, A. Ruiz Cortés, R. E. Lopez-Herrejon, and A. Egyed. 2016. Multi-Objective Test Case Prioritization in Highly-Configurable Systems: A Case Study. *Journal of Systems and Software* 122 (2016), 287–310.
[10] A. B. Sánchez, S. Segura, and A. Ruiz Cortés. 2015. SmarTest: Accelerating the detection of faults in Drupal. In *DrupalConEurope 2015*.
[11] A. B. Sánchez, S. Segura, J. A. Parejo, and A. Ruiz-Cortés. 2017. Variability Testing in the Wild: The Drupal Case Study. *Software and Systems Modeling Journal* 16, 1 (Apr 2017), 173–194.
[12] A. B. Sánchez, S. Segura, and A. Ruiz-Cortés. 2014. The Drupal Framework: A Case Study to Evaluate Variability Testing Techniques. In *Workshop on Variability Modelling of Software-intensive Systems*. ACM, 11:1–11:8.
[13] A. B. Sánchez, S. Segura, and A. Ruiz-Cortés. 2015. SmarTest: Proposal for accelerating the detection of faults in Drupal. In *DrupalCampSpain 2015*. Cadiz.
[14] T. Tomlinson and J. K. VanDyk. 2010. *Pro Drupal 7 development: third edition*.