Assessing the Test Suite of a Large System Based on Code Coverage, Efficiency and Uniqueness

László Vidács*, Ferenc Horváth†, Dávid Tengeri†, Árpád Beszédes†
*MTA-SZTE Research Group on Artificial Intelligence
University of Szeged, Szeged, Hungary
†Department of Software Engineering
University of Szeged, Szeged, Hungary
{lac, hferenc, dtengeri, beszedes}@inf.u-szeged.hu

***Abstract*—Regression test suites of evolving software systems play a key role in maintaining software quality throughout continuous changes. They need to be effective (in terms of detecting faults and helping their localization) and efficient (optimally sized and without redundancy) at the same time. However, test suite quality attributes are usually difficult to formalize and measure. In this paper, we rely on a recent approach for test suite assessment and improvement that utilizes code coverage information, but at a more detailed level, hence it adds further evaluation aspects derived from the coverage. The basic idea of the method is to decompose the test suite and the program code into coherent logical groups which are easier to analyze and understand. Several metrics are then computed from code coverage information to characterize the test suite and its constituents. We extend our previous study and employ derived coverage metrics (which express efficiency and uniqueness) to analyze the test suite of a large scale industrial open source system containing 27 000 test cases.**

***Keywords*-code coverage, regression testing, test suite evaluation, test suite quality, test efficiency, test metrics**

I. Introduction

The key player in regression testing is the *regression test suite*, which needs constant maintenance just as the software itself, otherwise its value will quickly decline [1], [2], [3], [4]. This typically includes the addition of new test cases and update or removal of outdated ones, after which it often becomes as large and complex as the software itself. Unfortunately, developers and testers have hardly any means that may help them in test suite maintenance activities, apart from perhaps test prioritization/selection and test suite reduction techniques [2], and some more recent approaches for the assessment of test code quality [5].

In earlier work [6], we introduced a method for a systematic assessment and improvement of test suites (named *Test Suite Assessment and Improvement Method – TAIME*), which is based on computing detailed code coverage information about the system and its test suite. This information is essentially a binary coverage matrix, where rows represent individual test cases while columns correspond to program elements such as statements or functions according to the chosen level of granularity. Both the test cases and the program elements are decomposed into coherent logical groups, which correspond to different *functional units* in the system. This way, various analyses can be performed on the coverage matrix – in addition to identifying low coverage areas –, such as identifying coverage patterns that indicate low coherence within functional units. Later, this approach was to measure the test suite of the WebKit project, a large industrially supported open source web browser layout engine [7]. For the analysis, we used the SoDA library [8].

In this paper, we provide results of a systematic evaluation of the data obtained for WebKit, and provide additional insights about this system and its test suite, primarily in terms of enhancement possibilities. WebKit has about 2.2 million lines of code and a large test suite of about 27 thousand test cases. Earlier, we identified 9 functional units in WebKit, and the evaluation was based on computing basic coverage metrics for these units. Results are summarized in a heat-map shown in Figure I. This visualization shows how different test groups cover different code groups in the system. The numbers in the cells represent code coverage ratios the test cases of a given test group attain with respect to the given code group (or to the whole system as indicated in the first row and column).

TABLE I
Coverage metric values and heat-map of WebKit

Test \ Code	WebKit	canvas	css	dom	editing	html5lib	http	js	svg	tables
WebKit	.53	.56	.61	.59	.67	.67	.65	.47	.50	.72
canvas	.16	.46	.26	.24	.07	.19	.00	.30	.03	.45
css	.24	.13	.51	.33	.25	.36	.00	.32	.11	.62
dom	.33	.17	.38	.52	.34	.51	.12	.35	.08	.57
editing	.23	.02	.31	.38	.66	.35	.01	.31	.06	.59
html5lib	.29	.12	.37	.43	.46	.52	.13	.34	.20	.63
http	.33	.23	.41	.42	.25	.41	.65	.39	.14	.57
js	.33	.16	.37	.47	.51	.44	.15	.44	.11	.63
svg	.26	.01	.38	.35	.17	.21	.01	.31	.50	.56
tables	.18	.00	.29	.30	.16	.31	.00	.26	.02	.62

In the present paper, basic coverage metrics are extended with efficiency and uniqueness metrics, and the functional units of the WebKit system are assessed in more detail. We make the following contributions:

- Detailed analysis of efficiency and uniqueness metrics trends computed on the 27 000 WebKit test cases.
- List of enhancement opportunities for each functional unit of the system.

II. Test Suite Assessment Metrics

In previous work [7], the analysis of the WebKit test suite was centered around the classical *code coverage* ratio (denoted by COV) and the so-called *partition* metrics (denoted by PART). The latter predicts the fault localization capability of a test suite because it captures how much the test cases are able to partition the program code regarding code coverage, which is important for separating faulty code from correct ones [7].

A certain degree of coverage or partitioning can be achieved using a different number of test cases. Clearly, the more test cases are in a suite, the better coverage and partition metrics are to be expected, provided the test cases are sufficiently different. However, if such test cases are added to the test suite, which mostly cover the same program code, they will unnecessarily increase the size of the test suite possibly with little additional benefit. Thus our assessment includes *efficiency* metrics of test suites, which take into account the relative number of test cases. To express efficiency we defined the following measures: *Coverage efficiency* (EFF_{COV}) shows how many procedures are covered by one test case on average; *Partitioning efficiency* (EFF_{PART}) is defined to express how much a single test contributes to the partitioning capability of the whole functional unit on average (a partition consists of procedures covered by the same test cases).

We are also interested in the *uniqueness* of the test groups in terms of how much their unique contribution is to the coverage of a code group compared to all other test cases in other test groups. To express this feature we defined two related metrics, *Specialization metric* (SPEC) and *Uniqueness metric* (UNIQ). As opposed to the earlier metrics, here we need information not only from the test and code group in question, but from other functional units as well. SPEC shows how specialized a test group is to a code group. A small SPEC value shows that there are relatively few test cases compared to all the others covering the given code group, while a high value reflects that there are few covering test cases outside of the given group.

The UNIQ metric measures what portion of the covered elements are covered only by a particular test group. A small UNIQ value shows that there are only a few procedures covered uniquely by the given test group and there are many other test cases covering the same. A high value indicates that the given test group is unique and there are few other tests covering the same procedures. The two uniqueness metrics are most useful to characterize tests which are designed to focus on certain code parts such as unit tests, or higher level functional tests concentrating on specific features. Integration tests typically exercise code from multiple modules by definition, hence they are probably less relevant from this aspect. More details on the aforementioned metrics can be found elsewhere [6], [7], [8].

Table II shows our extended list of metrics obtained for the WebKit system, more precisely for each of the 9 functional units identified. In the first section the number of procedures (at present, our analysis granularity is procedures), test cases and the *Tests per Procedure* (TPP) metric values are given. The second and third sections show efficiency metrics, while in the last section uniqueness metric values are presented.

TABLE II
SUMMARY OF BASIC AND EXTENDED METRICS FOR WEBKIT

	Statistics			Coverage efficiency		Partition efficiency		Uniqueness	
	Procs	*Tests*	*TPP*	*COV*	$\mathit{EFF}_{\mathit{COV}}$	*PART*	$\mathit{EFF}_{\mathit{PART}}$	*SPEC*	*UNIQ*
canvas	400	1073	2.68	0.46	0.17	0.69	0.26	0.77	0.36
css	1899	2956	1.56	0.51	0.33	0.72	0.46	0.12	0.09
dom	3761	3749	1.00	0.52	0.52	0.76	0.76	0.17	0.09
editing	1690	1358	0.80	0.66	0.82	0.87	1.08	0.06	0.16
html5lib	4176	2118	0.51	0.52	1.03	0.76	1.50	0.08	0.08
http	454	1778	3.92	0.65	0.17	0.79	0.20	0.5	0.61
js	8113	8313	1.02	0.44	0.43	0.68	0.66	0.37	0.10
svg	6336	1955	0.31	0.5	1.62	0.74	2.40	0.22	0.57
tables	2035	1340	0.66	0.62	0.94	0.83	1.26	0.06	0.03

III. Detailed Analysis of WebKit Metrics

A. Efficiency

Test efficiency has many facets such as execution time, defect detection rate, etc., which may be taken into consideration in general. In our case, we investigate the "cost" of achieving the base coverage metric values in terms of test case and procedure numbers. Since the same results can be achieved using various number of test cases, the same level of coverage/partitioning achieved by less test cases means that tests are more efficient on average in terms of code coverage.

The second section in Table II shows the values of coverage and coverage efficiency (higher values are better). Although the absolute coverage values of groups are similar, this ratio shows remarkable differences. For example, in `canvas` or `http` there are 0.17 covered procedures for a test on average; compared to `html5lib` or `svg`, where one test adds more than one covered procedure on average. The PART metric is shown together with the EFF_{PART} metric in the third section. The values can be examined similarly to coverage efficiency.

In a few cases high values of TPP need attention (`http`, `canvas`). The added value of individual test cases in these groups is smaller than in others. Test reduction is not necessarily required, but newly added tests could be more concentrated. On the other hand, these groups contain fewer test cases than the average. The two other efficiency values follow similar trends. The highest efficiency values are reached by the `svg` group, but its coverage is not as good as that of the `editing` and `tables` groups, which are well balanced: they reached coverage above the average, still their efficiency is good.

B. Uniqueness

While efficiency considers a standalone property of the test group, now the relation to other test groups is considered. We investigate to what extent tests cover procedures that are not covered by test cases of other functional units. The last

section in Table II presents uniqueness information. In order to interpret these results, also consider Figure 1, which shows the number of all test cases in each group, divided into special and not special parts (proportionally according to SPEC). In this regard, several test groups (`editing`, `tables`) could be improved. The test group of `canvas` has the best value with special test ratio of 0.77. It is followed by the `http` and the `js` groups, which latter contributes with the largest amount of special tests, and its SPEC value is still among the highest.

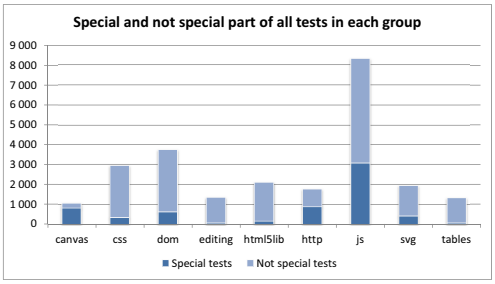

Fig. 1. Special and not special part of all tests in each group

The last column of the table reports UNIQ values. These values show what portion of the coverage metric is obtained uniquely by own test cases of groups. Figure 2 shows a bar-chart of the coverage metric, where the bars are divided into uniquely covered and commonly covered (*i.e.* also covered by other test groups) parts according to metric values of this last column. There are groups like `http`, `svg` and `canvas`, where UNIQ part represents a remarkable amount in the overall coverage of the test group. Not surprisingly, core groups (`css`, `dom`, `html5lib`, `js`) behave much worse, the majority of their coverage is not special, it is covered also by other groups as well. Except for integration tests, a common aim in improving test quality would be to increase the unique part of test groups. Developers also have to be careful in test case selection when uniqueness is high, because the chance to leave out a test that uniquely covers procedures is higher.

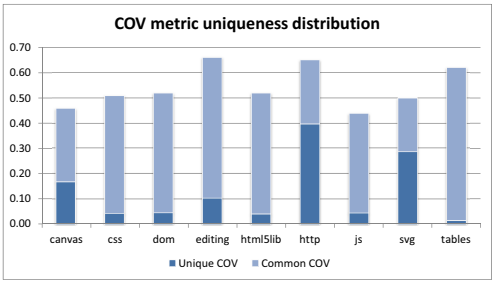

Fig. 2. Unique and common part of the COV metric for each group

C. Enhancement opportunities in WebKit

We summarize our observations in Table III, where each functional unit and the associated test group of the WebKit test suite is evaluated and a recommendation is provided. Note, that evaluation is based on the results of this paper extended with observations from [7] (which includes coverage and partition heat-maps). A general recommendation for the WebKit test suite could be that the coverage of all code groups could be improved as they are around the overall coverage rate of 53%. This should be done by adding more unique and specific test cases to the individual test groups. Another general comment is that component level testing (unit testing) is usually more effective than system level testing (as is the case with WebKit) when higher code coverage is aimed. For example, error handling code is very hard to be exercised during system level testing, while at component level such code can be more easily tested. Thus, in a long term, introducing a unit testing framework and adding real component tests would be beneficial to attain higher coverage.

IV. RELATED WORK

The main approach to assess the adequacy of testing has long been the fault detection capability of test processes [9]. Code coverage is a traditional base for white-box test design techniques due to the presumed relationship to defect detection capability, however this correlation is not always present or is at least not evident [10], [11]. We use the functional units as a priori information, and code coverage to gain more in-depth knowledge about the test suite and its relation to the system.

The area of test suite metrics is much less developed than general measurement for software quality. Athanasiou *et. al.* [5] gave an overview on the state of the art. They concluded that although some aspects of test quality had been addressed, basically it remained an open challenge. Researchers started to move towards test oriented metrics only recently, which strengthens our motives to work towards a more systematic evaluation method for testing. Gomez *et. al.* found that only a small fraction of metrics is directed towards testing [12]. Chernak [13] proposes that objective measures should be defined and built into the testing process to improve the overall quality of testing, but the employed measures in this work are also defect-based ones, as opposed to our paper.

V. CONCLUSIONS

Large and complex systems tend to grow large regression test suites as well. These test suites have to be effective in finding as many defects as possible, and efficient in terms of minimal redundancy to be useful on long term. In this paper we provided insights about the WebKit system and its test suite from this respect using our code coverage-based test suite assessment method. Over previous results, we added new dimensions to simple coverage-based analysis by computing efficiency and uniqueness metrics trends of code and test groups. The outcome of the metrics based analysis is a set of observations and a list of enhancement opportunities for the large scale WebKit test suite, which demonstrates the method's usefulness in the validation process of real life software tests.

Important future work will be to empirically evaluate our findings, and we also plan to investigate the possibilities of an automatic recommendation system that can at least partially automate the process of test suite assessment.

TABLE III
METRICS BASED EVALUATION OF WEBKIT TEST GROUPS

Group	Description	Evaluation
http	The http code base is responsible for testing the http protocol, the communication between the browser and the servers – assemble requests, send data, etc. Most functionalities are covered by the `http` test group, while other test groups usually use basic communication and small number of requests.	These groups have the highest UNIQ values, meaning that other test groups are not really exercising these code groups, while these test groups cover other code groups. Thus, if the test groups are to be modified, these should be preferred over the other groups. The `http` and `canvas` groups have the two highest TPP and lowest EFF_{COV} metric values, which is balanced by the two best SPEC properties. The number of test cases in these groups could probably be **reduced** without losing coverage, but only **with taking care of special tests**. The `svg` group has the highest EFF_{COV} and EFF_{PART} values, however, its COV could be **improved by new test cases**.
svg	Svg is a special format that allows the description of graphics using xml-like format. It can be embedded within a html document (similar to the canvas) or used as a separate document. The specific `svg` test group covers its code.	
canvas	Canvas is a special html element, an area where figures can be drawn (usually by some scripts). The `canvas` test group covers the canvas code group, while other tests do not really aim canvas.	
editing	This code group is responsible for various editing features of web page content like filling the input fields, support for text selection, copy-paste, etc. Some content manipulation features of JavaScript are also implemented through this codebase. As JS plays a central role in the tests, it implies that the `editing` code group has similar attributes.	These are the central elements of rendering a typical webpage. These functional units have low UNIQ values showing that other test groups provide notable coverage to these code groups. So the modification of these test groups is advised to carry out after those other groups. Although `js` contributes the highest number of special (SPEC) tests, its coverage value is the lowest among these groups. Thus, there is a room for improving it by **adding unique test cases** (to also improve the UNIQ value). The `editing` and `html5lib` groups have the highest coverage values. However, their own COV metric value is different. While `editing` tests provides almost the full coverage of its code group alone, a moderate part of the `html5lib` coverage is provided by other groups. The `editing` test group could probably be reduced by investigating and **removing test cases providing non-unique coverage**, while improving `html5lib` test group by **adding more unique tests** to it is probably a harder work. `css` and `dom` groups have similar metric values. The TPP value is lower for the `dom` group which implies higher efficiency metric values. In the case of `css`, the reduction of the specific test suite could probably be done by **removing test cases that are not unique** and similar to other test groups (by means of coverage).
js	WebKit tests are automated. The test inputs are specially prepared documents that are in connection with the test environment. The process is partially controlled from the test cases through JavaScript interfaces. As a result, all WebKit tests utilize JavaScript and cover a notable part of the `js` code, and only a small part of the procedures are uniquely covered by the `js` test group.	
css	In WebKit, all style information is handled through Cascading Style Sheets (css). Thus, similar to the `dom` or `html5lib`, the load of any document will imply the use of a large part of the `css` code base, which implies a weak uniqueness value.	
dom	For all documents, dom-trees are built in WebKit: whenever a page is loaded, a series of dom elements are created in order to allow dynamic content manipulation. As a consequence, all test groups cover the `dom` code group.	
html5-lib	The `html5lib` group code contains those classes in WebKit that will represent the different parts of the html documents. The content of the html documents are stored using these classes. As most of the WebKit test cases are html documents, most test groups cover this code group.	
tables	The `tables` code group is an outlier in the sense that this code group is heavily covered by all of the test groups. The reason for this is that it is hard to separate this group from the code implementing the so-called box model in WebKit, and the box model is used not only in the tables but it is a base of the rendering engine. Thus, almost anything that tests web pages will use the implementation of the box model, which is mostly included in the `table` code group.	`tables` maintains good COV and PART metrics, its coverage is the highest one while its SPEC and UNIQ values are the lowest. Highly covered by other test groups, `tables` should be the last one to be optimized among the test groups. The number of test cases in this group could probably be **reduced** due to the high coverage by other modules, however, **more specific tests** could be used to improve coverage.

REFERENCES

[1] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 85–103.

[2] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[3] K. Beck, Ed., *Test Driven Development: By Example*. Addison-Wesley Professional, 2002.

[4] L. S. Pinto, S. Sinha, and A. Orso, "Understanding myths and realities of test-suite evolution," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, pp. 33:1–33:11.

[5] D. Athanasiou, A. Nugroho, J. Visser, and A. Zaidman, "Test code quality and its relation to issue handling performance," *Software Engineering, IEEE Transactions*, vol. 40, no. 11, pp. 1100–1125, Nov 2014.

[6] D. Tengeri, Á. Beszédes, T. Gergely, L. Vidács, D. Havas, and T. Gyimóthy, "Beyond code coverage - an approach for test suite assessment and improvement," in *Proceedings of 2015 IEEE ICST Workshops (ICSTW), 10th Testing: Academic and Industrial Conference - Practice and Research Techniques (TAIC PART)*, Apr. 2015, pp. 1–7.

[7] F. Horváth, B. Vancsics, L. Vidács, Á. Beszédes, D. Tengeri, T. Gergely, and T. Gyimóthy, "Test suite evaluation using code coverage based metrics," in *Proceedings of the 14th Symposium on Programming Languages and Software Tools (SPLST'15)*, Oct. 2015, pp. 46–60.

[8] D. Tengeri, Á. Beszédes, D. Havas, and T. Gyimóthy, "Toolset and program repository for code coverage-based test suite analysis and manipulation," in *Proc. of the IEEE Intl Working Conference on Source Code Analysis and Manipulation (SCAM'14)*, Sep. 2014, pp. 47–52.

[9] H. Zhu, P. A. V. Hall, and J. H. R. May, "Software unit test coverage and adequacy," *ACM Comput. Surv.*, vol. 29, no. 4, pp. 366–427, Dec. 1997.

[10] A. S. Namin and J. H. Andrews, "The influence of size and coverage on test suite effectiveness," in *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis*. ACM, 2009, pp. 57–68.

[11] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 435–445.

[12] O. Gómez, H. Oktaba, M. Piattini, and F. García, "A systematic review measurement in software engineering: State-of-the-art in measures," in *Software and Data Technologies*, ser. Communications in Computer and Information Science. Springer, 2008, vol. 10, pp. 165–176.

[13] Y. Chernak, "Validating and improving test-case effectiveness," *IEEE Softw.*, vol. 18, no. 1, pp. 81–86, Jan. 2001.