Augmenting Test Suites Effectiveness by Increasing Output Diversity

Nadia Alshahwan and Mark Harman
CREST Centre
University College London
Malet Place, London, WC1E 6BT, U.K.
{*nadia.alshahwan.10,mark.harman*}*@ucl.ac.uk*

***Abstract*—The uniqueness (or otherwise) of test outputs ought to have a bearing on test effectiveness, yet it has not previously been studied. In this paper we introduce a novel test suite adequacy criterion based on output uniqueness. We propose 4 definitions of output uniqueness with varying degrees of strictness. We present a preliminary evaluation for web application testing that confirms that output uniqueness enhances fault-finding effectiveness. The approach outperforms random augmentation in fault finding ability by an overall average of 280% in 5 medium sized, real world web applications.**

***Keywords*-SBSE; HTML output; Web applications**

I. Introduction

Why does structural coverage remain so prevalent as the only adequacy criterion considered in so many studies of software testing? This paper argues that we need more research on complementary criteria and proposes one based on uniqueness. Although higher coverage may increase fault detection, there is much controversy about coverage being the only contributing factor [6]. We propose a novel criterion that is based on the uniqueness of the program's output to enhance traditional coverage criteria. We expect that raising the diversity of the output could lead to test suites that are more effective at exposing faults.

Faults with high severity often propagate to the observable output and affect user perception of an application [5]. Therefore, the output provides a valuable resource for identifying unexpected behaviour that is more critical from a user's point of view. In applications with rich outputs such as web applications, the output provides a more valuable resource for our proposed criterion. The complexity and richness of the output may also make it more likely for faults to propagate to the output and for the approach to be effective. This is the proposition upon which this paper reports.

In this paper we apply our uniqueness approach to augment test suites that were created using traditional structural coverage criterion (branch coverage) with test cases that provide unique outputs. We report results for 4 output uniqueness definitions applied to web applications. The contributions of this paper are as follows:

- The introduction of a novel test adequacy criterion based on output uniqueness.
- Four definitions of output uniqueness in the context of web applications.
- An initial evaluation on 5 real world web applications of the 4 output uniqueness definitions in terms of their ability to find real faults. The initial results indicate that output uniqueness is a valuable criterion for generating test suites that are more effective at exposing faults. Uniqueness outperforms random augmentation by an average of 280% over all applications.

II. Approach

Checking the output of a test case is a comparatively cheap operation: It neither requires instrumentation of the code nor the building of intermediate structures, such as control flow graphs or data flow models. This makes it easy to experiment with a large number of test cases to find the ones that provide interesting or 'unique' outputs.

We propose that output uniqueness shall be used to augment a traditional test generation adequacy criterion not replace it; output alone would not capture all faults that can manifest in an application. For example, two different errors that cause a web application to return a blank page would have the same output. However, as we shall show in this paper, focusing on output uniqueness improves fault finding ability, even for relatively high coverage test suites.

In this paper we use output uniqueness to augment a test suite generated to satisfy a traditional test adequacy criterion (branch coverage). In the remainder of this section we first discuss web application output and propose 4 definitions of client–side page output uniqueness. We then describe how we propose to use these definitions in test suite augmentation.

A. Web Application Output

The principle output of a web application, visible to the user, is a client–side HTML page. The client–side page can be either static or dynamically generated based on user choices when the page was requested. The page is composed of the content, the HTML code and embedded elements such as images or client–side scripts (e.g. JavaScript). In this paper we focus on the 2 main elements of the client–side page:

The Content: The content (C) is the textual data that is presented to the user. The main element in this content is the data that the user requested. For example in case of a search, the content is the list of matching items.

However, extra helping text can be found throughout the output page. Examples of such text are page titles, welcome messages and field labels.

HTML Code: HTML code (H) defines how the page is presented to the user. The content is organized in tables or frames that are constructed using HTML. Aesthetic elements such as colours and fonts as well as backgrounds and embedded pictures can also be specified using HTML tags. In addition to how the page looks, HTML can be used to define functional elements that enable the user to interact with the application. HTML Forms and links are the main examples of these elements. HTML code is constructed of HTML tags (T) that define the type of the element (e.g. table, form, input). Each tag has a number of relevant attributes (A) such as actions for forms or values for input.

We can define a web applications client–side page output as a tuple $O =< C, H >$.

Algorithm 1 Test Data Generation Algorithm: Starting from a test suite the algorithm generates 4 test suites that each satisfy one of the output uniqueness definitions

Require: Test Suite *TS*
1: **for all** T in *TS* **do**
2: *output* = executeTestCase(T)
3: $\mathcal{O} = \mathcal{O} \cup$ *output*
4: **end for**
5: **for all** T in *TS* **do**
6: **while** Number of tries $<= 100$ **do**
7: $T' =$ mutateInput(T)
8: *output* = executeTestCase(T')
9: **if** OU-AllSatisfied($\mathcal{O}$,*output*) **then**
10: *TS-ALL* = *TS-ALL* $\cup T'$
11: **if** OU-TextSatisfied($\mathcal{O}$,*output*) **then**
12: *TS-Text* = *TS-Text* $\cup T'$
13: **end if**
14: **if** OU-StructSatisfied($\mathcal{O}$,*output*) **then**
15: *TS-Struct* = *TS-Struct* $\cup T'$
16: **end if**
17: **if** OU-SeqSatisfied($\mathcal{O}$,*output*) **then**
18: *TS-Seq* = *TS-Seq* $\cup T'$
19: **end if**
20: **end if**
21: $\mathcal{O} = \mathcal{O} \cup$ *output*
22: **end while**
23: **end for**
24: **return** *TS-All*, *TS-Text*, *TS-Struct*, *TS-Seq*

B. Client–side Page Output Uniqueness

To apply our approach we need to first define 'uniqueness'. A strict definition of output uniqueness could capture all generated test cases that cause a fault that propagates to the output. However, a strict definition could also lead to an explosion in the test suite size, with many additional test cases that may yield no additional benefit.

Our aim is to evaluate a number of output uniqueness definitions to identify the most effective definition that captures interesting output differences while maintaining a smaller test suite. We define a test suite as a set of (input,output) pairs. We first consider the strictest definition:

Definition 1. *Output o is OU-All unique with regard to a test suite T $\iff$ for all (i, o') there exists at least one observable difference between o and o'.*

When a new output page is analysed, any difference in any element of the page compared to all previously visited pages categorizes the new output page as unique.

This definition could potentially lead, in some cases, to infinitely many unique outputs that do not necessarily enhance the test suite's effectiveness, but considerably increase the oracle cost. For example, an application that displays the date on the output page could result in a potentially infinite set of unique outputs. A page that displays product information would have as many unique outputs as there are products in its database. To overcome this problem we can also define output uniqueness, less strictly, in terms of the HTML structure of the page ignoring the text.

Definition 2. *Output o is OU-Struct unique with regard to a test suite T $\iff$ for all (i, o') where $o =< c, h >$ and $o' =< c', h' >$ there exists at least one observable difference between h and h'.*

This definition eliminates the 'potentially infinite output' issue in the text discussed for OU-All. However, the HTML structure may still yield large test suites. Consider the product pages of items again, if the form to order an item contains a hidden field that holds the item's ID, we will have as many unique outputs as there are products in the database. We add a new definition of output uniqueness where the HTML structure of a page is stripped of any text or embedded values and only the opening and closing tags are considered. This is to eliminate any variations that are caused by form options, default values or font and style settings.

Definition 3. *Output o is OU-Seq unique with regard to a test suite T $\iff$ for all (i, o') where $o =< c, h >$ and $o' =< c', h' >$ and where h and h' contain a set of tags t and t' and attributes a and a' respectively and there exists at least one observable difference between t and t'.*

The previous two definitions focused on the HTML structure of a page. However, the text in the page can contain error messages produced by the server. Therefore, we add a final definition of output uniqueness:

Definition 4. *Output o is OU-Text unique with regard to a test suite T $\iff$ for all (i, o') where $o =< c, h >$ and $o' =< c', h' >$ there exists at least one observable difference between c and c'.*

C. Generating Test Cases for Output Uniqueness

Algorithm 1 describes how we use output uniqueness to generate new test cases. The algorithm takes a test suite

as an input and builds new test suites that satisfy each of the output uniqueness definitions in Section II-B. For each original test case, one input is mutated at a time to generate a new test case. The input is mutated, with equal probability, by either assigning a random value or a value collected dynamically from the output of the original test suite. The new mutated test case is then executed and the output is examined to determine which output uniqueness definition it satisfies. For each definition a test suite is maintained that contains all mutated test cases that satisfy its definition.

III. Evaluation

We designed the evaluation to answer the following research questions:

RQ1: Does using output-uniqueness augmented test suites enhance fault finding ability?

RQ2: How do the 4 definitions of output uniqueness affect the fault finding ability and test effort of the generated test suite?

To answer these 2 questions we augment a test suite generated using a tool called SWAT from previous work [1] with test cases that enhance output uniqueness for each of the 4 definitions. We then compare the fault finding ability of the original suite to the augmented suites and also to the original SWAT test suite augmented by the same number of additional test cases selected randomly. We calculate and compare the number of faults per new test case of each of the definitions compared to random. We perform Wilcoxon paired one-sided signed rank test at the 95% confidence level to determine the statistical significance of the observed results. To answer **RQ2** we compare the fault finding ability and sizes of the new test suites for the 4 definitions. Test suite size is only an approximation of test effort. A larger test suite would require more time and effort to execute, maintain and examine the output.

Table I
The web applications used in the study

App Name	Version	PHP Files	PHP ELoC	Description
FAQForge	1.3.2	19	834	FAQ management tool
Schoolmate	1.5.4	63	3,072	School admin system
Webchess	0.9.0	24	2,701	Online chess game
PHPSysInfo	2.5.3	73	9,533	System monitoring tool
Timeclock	1.0.3	62	14,980	Employee time tracker

Experimental Set-up: Table I provides information about the 5 applications we used in our evaluation. We used the test suite with the highest branch coverage from 30 test suites previously generated by SWAT [1] as an input to a new tool SWAT-U that implements Algorithm 1. We chose the highest coverage because we want to focus on additional fault finding ability of uniqueness, even where coverage is relatively high. We also used the same SWAT test suite and mutation algorithm for random augmentation. Since the test data generation process is non-deterministic, we run SWAT-U and random augmentation 30 times for each test suite, to support statistical significance testing.

Table II
Results of average faults found and test suite size obtained from running the approach and random 30 times for each application. The (%) column in New Faults is calculated in relation to original faults found. Faults/Test results in bold perform statistically significantly better than random.

App Name	Original Cov	Original Tests	Original Faults	Algorithm	New Tests	New Faults Num	New Faults %	Faults /Test	Improv. on Rand
FAQForge	69%	36	50	Rand	180	3.6	7.1	0.020	-
				OU-All	180	5.0	9.9	**0.028**	40%
				OU-Text	96	1.3	2.7	0.014	-30%
				OU-Struct	170	5.0	9.9	**0.029**	48%
				OU-Seq	3	0.8	1.6	0.245	1137%
Schoolmate	70%	176	103	Rand	314	1.7	1.6	0.005	-
				OU-All	314	23.7	23.0	**0.076**	1324%
				OU-Text	120	21.4	20.8	**0.178**	3255%
				OU-Struct	252	20.3	19.7	**0.080**	1415%
				OU-Seq	32	9.9	9.6	**0.309**	5737%
Webchess	33%	49	70	Rand	299	0	0	0	-
				OU-All	299	0	0	0	0%
				OU-Text	54	0	0	0	0%
				OU-Struct	220	0	0	0	0%
				OU-Seq	2	0	0	0	0%
PHPSysInfo	22%	20	7	Rand	1138	2.1	30.0	0.002	-
				OU-All	1138	2.1	30.0	0.002	0%
				OU-Text	1138	2.1	30.0	0.002	0%
				OU-Struct	228	1.6	22.9	**0.007**	289%
				OU-Seq	8	1.7	24.8	**0.228**	12567%
Timeclock	21%	284	172	Rand	1366	2.7	1.6	0.002	-
				OU-All	1366	3.7	2.2	**0.003**	37%
				OU-Text	98	3.7	2.2	**0.038**	1812%
				OU-Struct	1317	2.7	1.6	0.002	2%
				OU-Seq	9	2.7	1.6	**0.307**	15408%

We use a fully automated oracle to detect faults. Our oracle parses PHP error log files and the output HTML pages of each test case for faults. Only faults that are caused by a unique code location and have a distinct type are counted.

We use database tables to keep track of previous output. Execution times of SWAT-U are not reported in this paper but the slowest run we encountered was 20 minutes.

Results: Table II reports the results of running the approach 30 times on each application together with information about the original test suites. The new test cases increased coverage only by 1% or less for all applications with no additional coverage for FaqForge. The last column reports the improvement in percentage of each algorithm over random. This is calculated based on the faults per test case.

In 3 of the 5 applications OU-All performed better than random augmentation. In Webchess no new faults were found by either the uniqueness approaches or random. This could be because no other faults exist or due to a limitation in the mutation algorithm. PHPSysInfo only has 4 user inputs which could limit the effect of user inputs on outputs.

As expected OU-All adds the largest number of test cases to the test suite while OU-Seq adds the fewest. OU-Struct and OU-Text perform differently based on the application. In both PHPSysInfo and Timeclock, the number of unique outputs based on the strictest definition OU-All yield a comparatively large number of new test cases. Both of these applications contain a time-dependent element in the

output: PHPSysInfo prints the time the system has been available while Timeclock prints weather information. This reinforces our arguments for the need for different output uniqueness definitions.

OU-All also finds the largest number of faults with a few exceptions where another definition finds a matching number of faults with fewer test cases. In all applications OU-Seq is the most efficient definition that found the largest number of faults per test case.

Answers to Research Questions: In this section we answer the research questions we posed at the start of this section.

RQ1: Does using output-uniqueness augmented test suites enhance fault finding ability? In 4 of the 5 applications studied at least one output uniqueness definition performed better than random. In 3 applications OU-All performed statistically significantly better than random. The best performing uniqueness definition significantly outperforms random augmentation in 3 of 5 applications by an overall average of 6,970% with one application (Webchess) where improvement was not possible.

RQ2: How do the 4 definitions of output uniqueness affect the fault finding ability and test effort of the generated test suite? OU-All found the most faults overall but adds the largest number of test cases. Less strict definitions result, in most cases, in a loss of some of the faults found by OU-All. OU-Seq performed the best in terms of faults per additional test case and shows the most potential.

IV. Related Work

In web application testing, traditional test adequacy criteria such as statement [3] and branch [1] coverage have been used for generating test suites. A new definition of these criteria has also been applied [8] where web pages, instead of statements, are the nodes in the control flow graph, while links are the edges. The output uniqueness criterion proposed in this paper aims to augment these criteria rather than to replace them.

Raghavan and Garcia-Molina [7] developed a crawler that extracts data behind web forms and therefore needs to identify pages that return 'errors' such as 'no records found'. The output is analysed to identify pages that appear frequently in response to a form submission and mark them as error pages. In our paper we aim to find pages where an execution error has occurred.

Sprenkle et al. [9] used the HTML output to automate the oracle in regression testing while Di Lucca [4] used it to detect clones in static web pages. Our implementation uses some concepts of categorizing HTML output and resolving issues in dynamic content that these previous research papers used. However, we apply those concepts to test case generation rather than oracle automation and clone detection.

Artzi et al. [2] used a path constraint similarity criterion to generate test cases to localize faults with minimal test cases. Their approach generates additional test cases for previously known faults, while we generate test cases to find new faults.

V. Conclusion and Future Work

In this paper we propose a new test generation criterion that is based on output uniqueness. In our preliminary evaluation, this new criterion proved to be useful in finding test cases that can reveal new faults. The results of the 4 definitions show that a less strict definition based on the HTML tags of a page was the most effective.

This new criterion poses new open research questions to further evaluate and understand how output uniqueness affects fault finding ability. One question is how does the starting test suite, in terms of coverage or fault finding ability, affect results. Also: How can we define output uniqueness for applications with less complex outputs? For example, for a program that outputs numbers, output uniqueness can be defined in terms of ranges or data types. Can we combine coverage and output uniqueness in one test data generation tool? What is the optimum point in the generation process to switch from coverage to output uniqueness criteria?

For web applications, we can use the results of this study to define a new criterion that decomposes and mixes elements of the different client–page components to achieve higher effectiveness. We can also include other elements of the output in our definitions such as the application state.

Output uniqueness can also be used as an alternative criterion when all or part of the application code is unavailable for instrumentation as, for example, in cases where an application uses third party components. This also suggests that relationships between white–box coverage and output uniqueness may prove interesting in future work.

References

[1] Nadia Alshahwan and Mark Harman. Automated web application testing using search based software engineering. In *ASE '11*, 2011.

[2] Shay Artzi, Julian Dolby, Frank Tip, and Marco Pistoia. Directed test generation for effective fault localization. In *ISSTA '10*, pages 49–60, 2010.

[3] Shay Artzi, Adam Kiezun, Julian Dolby, Frank Tip, Daniel Dig, Amit Paradkar, and Michael D. Ernst. Finding bugs in web applications using dynamic test generation and explicit-state model checking. *IEEE Trans. Softw. Eng.*, 36:474–494, 2010.

[4] Giuseppe A. Di Lucca, Massimiliano Di Penta, and Anna Rita Fasolino. An approach to identify duplicated web pages. In *COMPSAC '02*, pages 481–486, 2002.

[5] Kinga Dobolyi and Westley Weimer. Modeling consumer-perceived web application fault severities for testing. In *ISSTA '10*, pages 97–106, 2010.

[6] Akbar Siami Namin and James H. Andrews. The influence of size and coverage on test suite effectiveness. In *ISSTA '09*, pages 57–68, 2009.

[7] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. In *VLDB '01*, pages 129–138, 2001.

[8] Filippo Ricca and Paolo Tonella. Analysis and testing of web applications. In *ICSE '01*, pages 25–34, 2001.

[9] Sara Sprenkle, Emily Gibson, Sreedevi Sampath, and Lori Pollock. Automated replay and failure detection for web applications. In *ASE '05*, pages 253–262, 2005.