Coverage and Fault Detection of the Output-Uniqueness Test Selection Criteria

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

ABSTRACT

This paper studies the whitebox coverage and fault detection achieved by Output Uniqueness, a newly proposed blackbox test criterion, using 6 web applications. We find that output uniqueness exhibits average correlation coefficients of 0.85, 0.83 and 0.97 with statement, branch and path coverage respectively. More interestingly, output uniqueness finds 92% of the real faults found by branch coverage (and a further 47% that remained undetected by such whitebox techniques). These results suggest that output uniqueness may provide a useful surrogate when whitebox techniques are inapplicable and an effective complement where they are.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Reliability, Verification

Keywords

Software Testing, Blackbox testing, Whitebox testing, Web applications

1. INTRODUCTION

This paper studies the blackbox test selection criterion output uniqueness, which was introduced in our recent NIER paper [5]. Output uniqueness is a purely blackbox technique that maximises differences between observed outputs. In the paper we report on the degree to which output uniqueness can complement whitebox criteria and act as viable surrogate. Specifically, we study the correlations of seven definitions of uniqueness with more well-established, whitebox criteria: statement, branch and path coverage. We also study and compare fault detection capability for output uniqueness and these structural whitebox coverage criteria.

We study output uniqueness for 6 web applications. We chose to study web applications because of their structurally rich output, which supports investigation of many different definitions of 'uniqueness'. Our definitions range from the almost implausibly permissive (differences in all but the last line are ignored) to the least permissive possible (every character counts). In-between these extremes, we study a hierarchy of subsuming definitions of uniqueness.

Our study is partly motivated by the many previous studies [6, 16, 25, 29] that have reported that achieving high structural whitebox coverage may leave many faults undetected. This previous work indicates the need for additional (and perhaps complementary) notions of test adequacy and coverage. Our results do, indeed, indicate that output uniqueness finds many of the faults found by whitebox coverage and also complements whitebox techniques by finding many faults that it leaves undetected.

The key insight that underpins the notion of output uniqueness is that two test cases that yield different kinds of output may likely traverse two different paths. Moreover, in systems with rich and structured outputs (such as, but not limited to, web applications), we expect faults to be more likely to propagate to the output and thereby produce a different output. Clearly, these observations depend on the definition of 'output difference' (which we define more rigorously later).

We use seven different definitions of output difference to select those test inputs that yield unique outputs (outputs that no other test input produces). Four of these definitions appear in our previous NIER paper. In the present paper we introduce Three new definitions and investigate our conjecture that output unique test suites would be likely to achieve high structural coverage, even though they were created with no knowledge of structure. We also investigate the conjecture that output unique test suites would likely enjoy high fault finding potential.

The results of our empirical study provide evidence to support these two conjectures. Specifically, we observe average Spearman rank correlations of 0.85, 0.83 and 0.97 respectively for statement, branch and path coverage with our strictest criterion and strong correlations for less strict criteria. This suggests that output uniqueness, though it is purely blackbox, is closely correlated to whitebox coverage.

We also report on the faults found by both output uniqueness and by whitebox techniques. Our results reveal that output uniqueness is good at finding real faults in the systems to which it is applied, consistently finding 92% of the faults found by branch coverage and a further 47% of additional faults that were not found by branch coverage.

The primary contributions of the paper are as follows:

1. An empirical investigation of the correlation between output uniqueness and structural coverage of test suites on six real-world web applications.
2. An empirical investigation of the correlation between output uniqueness and fault detection capabilities of test suites.
3. An empirical investigation of the consistency of output uniqueness criteria in finding faults and their complementarity to branch coverage.

The findings of these studies indicate that output uniqueness can be used as both a surrogate and a complement to whitebox techniques. This offers hope to testers working in situations where whitebox techniques are difficult to apply. For example, where non-functional properties are to be tested, such as execution time, insertion of the instrumentation needed by whitebox testing will disrupt the property that is under test [1]. For test engineers tasked with testing third party code, the source may not be available and so whitebox techniques would be inapplicable. Output uniqueness may provide a useful surrogate in these situations as well as being a supplementary technique in cases where whitebox testing is applicable.

In the rest of this paper, Section 2 presents our proposed output uniqueness criteria, whilst Section 3 presents the empirical studies together with a discussion of their results, actionable findings and threats to validity. Section 4 presents related work and Section 5 concludes.

2. OUTPUT UNIQUENESS CRITERIA

To utilise output uniqueness as a selection criterion, we first need to define what constitutes a unique output. The authors' NIER paper [5] introduced the concept of Output Uniqueness (OU), presenting four different OU criteria (OU-All, OU-Text, OU-Struct and OU-Seq). In this section we briefly review these four criteria, to make the paper self contained, and introduce three new criteria, yielding a family of seven OU criteria for blackbox testing.

2.1 Web Application Output

Web output consists of the HTML code (H) and content (C): The content (C) is the textual data visible to the user. The HTML code (H) defines the structure and appearance of the page as well as elements, such as Forms and links.

The HTML code consists of nested HTML tags (T) that define the element type (e.g. table, input). Each tag $t \in T$ has a set of attributes (A). For example, a tag t of type input could have attributes a_{value} and a_{type}. An application's client-side output page is defined as a tuple $O = \langle C, H \rangle$, where the HTML code H is a set T of tags t and each tag in T is associated with a set A of attributes a.

2.2 OU Definitions

The client-side output in Figure 1(a) is taken from one of the applications studied in the paper (Schoolmate) and simplified for readability. This code will be used to illustrate those parts of the output that are considered significant for the purposes of defining each output uniqueness criterion.

Five of our OU criteria are based on the HTML structure, while a further two are based on the content. A test suite is defined as a set of (input, output) pairs. The strictest structure based definition to consider is:

Definition 1 Output o is OU-All unique with regard to a test suite $T \iff \forall\ (i, o') \in T\ o \neq o'$.

When a new output page is compared to previously visited pages, any difference in the page categorises the new output page as unique. All the HTML code in Figure 1(a) will be considered for comparison when using OU-All.

This definition could potentially lead, in some cases, to arbitrarily many unique outputs that do not necessarily enhance the test suite's effectiveness, but considerably increase oracle costs. 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 output uniqueness can be defined, less strictly, in terms of the HTML structure of the page (ignoring the textual content).

Definition 2 Output o is OU-Struct unique with regard to a test suite $T \iff \forall\ (i, o') \in T$ where $o = \langle C, H \rangle$ and $o' = \langle C', H' \rangle\ H \neq H'$.

Figure 1(b) shows the part of the output page that will be considered for comparison when using OU-Struct. The text in the output page is removed and only the HTML structure is retained and compared to previously observed output to decide whether the output is new.

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, for example, when the structure embeds context-aware advertisements. We, therefore, provide a new definition of output uniqueness that retains input names and considers only input values for fields of hidden type. This is to eliminate any variations that are caused by Form options, default values or style settings.

Hidden Form variables are control variables that are embedded by the server-side code in Forms to pass state information to the client-side. Their role as conduits for state information means that hidden Form variables may be expected to be significant in capturing application behaviour. Unexpected values held in hidden Form variables can possibly indicate a fault in a previous execution.

Definition 3 Output o is OU-Hidden unique with regard to a test suite $T \iff \forall\ (i, o') \in T$ where $o = \langle C, H \rangle$ and $o' = \langle C', H' \rangle$ and H and H' contain a set of tags T and T' and attributes A and A' respectively, either $T \neq T'$ or $t_i \in T$ and $t'_i \in T'$ are equal to 'input' and either $a_{name_i} \neq a'_{name_i}$ or a_{type_i} and a'_{type_i} are equal to 'hidden' and $a_{value_i} \neq a'_{value_i}$.

The subscripts to attributes a denote the type of attribute being considered. An attribute a_{name_i} is an attribute of type name in the set of attributes A associated with the ith element in the set of tags T. Figure 1(c) shows that part of the output that will be considered for OU-Hidden.

Hidden Form variables can also lead to arbitrarily large unique output sets. For example, suppose the item order Form for an online store contains a hidden field that holds the item's ID. There will be as many OU-Hidden unique outputs as there are products in the database. Therefore, it might be useful to consider characteristics of hidden Form variable values rather than their actual values.

A new definition is, therefore, proposed based on the subtypes of hidden Form variable values. The subtypes used in this paper are: positive and negative numbers, strings, zeros, empty strings and NULL. These subtypes are chosen to be general (not application-specific) in order to avoid experimenter bias in our empirical studies.

```
<html>
 <head>
 <title>SchoolMate
    </title>
 </head>
 <body>
 <form action='index.php'
   method='post' name='login'>
 Username:
 <input type=text name='user'>
 Password:
 <input type=password name='pw'>
 <input type=submit value='Log'>
 <input type=hidden name='page'
       value='1'>
 </form>
 <span class='footer'>Powered By
                -SchoolMate</a>
 </body>
</html>
```

(a) OU-All

```
<html>
 <head>
 <title>
    </title>
 </head>
 <body>
 <form action='index.php'
   method='post' name='login'>

 <input type=text name='user'>

 <input type=password name='pw'>
 <input type=submit value='Log'>
 <input type=hidden name='page'
     value='1'>
 </form>
 <span class='footer'>
                             </a>
 </body>
</html>
```

(b) OU-Struct

```
<html>
 <head>
 <title>
    </title>
 </head>
 <body>
 <form>


 <input name='user'>

 <input name='pw'>
 <input>
 <input name='page'
     value='1'>
 </form>
 <span>
       </a>
 </body>
</html>
```

(c) OU-Hidden

```
<html>
 <head>
 <title>
    </title>
 </head>
 <body>
 <form>


 <input name='user'>

 <input name='pw'>
 <input>
 <input name='page'
    value=num>
 </form>
 <span>
       </a>
 </body>
</html>
```

(d) OU-Subtypes

```
<html>
 <head>
 <title>
   </title>
 </head>
 <body>
 <form>


 <input>

 <input>
 <input>
 <input>

 </form>
 <span>
       </a>
 </body>
</html>
```

(e) OU-Seq

Figure 1: Output used by each structure based output uniqueness criteria on a simplified output page from the Schoolmate application: OU-All considers all the output, OU-Struct considers only HTML structure, OU-Seq considers HTML tags, but ignores attributes, OU-Hidden ignores all attributes except input names and hidden fields, OU-Subtypes considers only the type of hidden fields.

However, in practice, the tester might define more specific subtypes drawn from the domain of the application under test (or for classes of such applications). This may allow the tester to better incorporate domain knowledge into the testing process.

Definition 4 Output o is OU-Subtypes unique with regard to a test suite $T \iff \forall\ (i, o') \in T$ where $o = \langle C, H\rangle$ and $o' = \langle C', H'\rangle$ and H and H' contain a set of tags T and T' and attributes A and A' respectively, either $T \neq T'$ or $t_i \in T$ and $t'_i \in T'$ are equal to 'input' and either $a_{name_i} \neq a'_{name_i}$ or a_{type_i} and a'_{type_i} are equal to 'hidden' and the subtype of $a_{value_i} \neq$ subtype of a'_{value_i}.

Figure 1(d) shows the part of the output that will be considered for OU-Subtypes. The value of the first hidden field `page` was replaced by the corresponding subtype 'num'.

A final structure based definition of output uniqueness can be proposed where the HTML structure of a page is stripped of any text or embedded values, including hidden field values, and only the opening and closing tags are considered:

Definition 5 Output o is OU-Seq unique with regard to a test suite $T \iff \forall\ (i, o') \in T$ where $o = \langle C, H\rangle$ and $o' = \langle C', H'\rangle$ and where H and H' contain a set of tags T and T' and attributes A and A' respectively $T \neq T'$.

Figure 1(e) shows the part of the output that will be considered for OU-Seq. All text and attributes from HTML tags are removed.

The previous five definitions focused on the HTML structure of an output page. However, the text in the page produced by the server may also be important. Therefore, a new definition of output uniqueness is added:

Definition 6 Output o is OU-Text unique with regard to a test suite $T \iff \forall\ (i, o') \in T$ where $o = \langle C, H\rangle$ and $o' = \langle C', H'\rangle$ $C \neq C'$.

For the HTML example in Figure 1(a), only the following parts of the output page will be considered:

```
SchoolMate
  Username:
  Password:
Powered By-SchoolMate
```

OU-Text only considers the text in the output page.

The last line of this text may be useful if, for example, it flags a failure or denotes the final result of a computation. Of course, it might also be the case that the last line of output text is always identical ('all completed successfully' or some such). If it turns out to be effective at finding faults then checking only the last line of text will certainly simplify the tester's job should he or she be playing the role of oracle [22]. We will, therefore, investigate the use of OU-LastText as a criterion to investigate whether it is sufficiently discriminating to be used as a useful blackbox criterion:

$$\begin{tikzcd}
 & \text{OU-All} \arrow[dl] \arrow[dr] & \\
\text{OU-Text} \arrow[d] & & \text{OU-Struct} \arrow[d] \\
\text{OU-LastText} & & \text{OU-Hidden} \arrow[d] \\
 & & \text{OU-Subtypes} \arrow[d] \\
 & & \text{OU-Seq}
\end{tikzcd}$$

Figure 2: Uniqueness criteria strictness hierarchy.

Definition 7 Output o is OU-LastText unique with regard to a test suite $T \iff \forall\ (i, o') \in T$ where $o = \langle C, H\rangle$ and $o' = \langle C', H'\rangle$ and c_{last} and c'_{last} are the last lines of C and C' respectively, $c_{last} \neq c'_{last}$.

For the HTML example in Figure 1(a) only the last line '`Powered By-SchoolMate`' would be considered for comparison by the OU-LastText criterion. In this case, such a last line of text would be an example of a situation in which OU-LastText is likely to be unhelpful; all HTML output pages contain the terminating sentence '`Powered By-SchoolMate`'. However, in other situations the last line may be better at distinguishing the type of computation performed.

Figure 2 illustrates the strictness hierarchy of our proposed output uniqueness criteria: OU-All is the strictest criterion, while OU-LastText and OU-Seq are the least strict. Structure based criteria (right branch) and text based criteria (left branch) are orthogonal. The OU relations form a strictness hierarchy: $A \to B$ means that A is stricter than B. That is, if B reports two outputs as unique, then A will agree that they are unique, but not necessarily vice versa.

3. EMPIRICAL STUDY

This empirical study is designed to answer the following four research questions:

RQ1: Structural Coverage Ability: How do the output uniqueness criteria correlate to whitebox coverage?

This research question investigates how well our output uniqueness correlates to statement, branch and path coverage criteria. A strong correlation indicates that output uniqueness criteria can be used as alternative criteria when these whitebox criteria are found to be inapplicable (e.g. when code is unavailable).

RQ2: Fault Finding Ability: How do the output uniqueness criteria correlate to fault finding ability of a test suite?

If a correlation is found in answer to RQ1, then this would mean that output uniqueness *could* be used in place of structural coverage, but whether it *should* be used depends on its ability to find faults. Therefore, RQ2 investigates the correlation between output uniqueness criteria and fault finding ability. It also examines how output uniqueness criteria compare to structural coverage criteria in their correlation to fault detection. To determine the usefulness of output uniqueness criteria, a strong correlation should exist between the number of unique outputs in a test suite and the number of faults found by the test suite.

RQ3: Fault Finding Consistency: How does output uniqueness consistency compare to structural coverage?

If RQ1 demonstrates that output uniqueness is correlated to whitebox coverage and RQ2 indicates a strong correlation with fault finding then output uniqueness should be used as a way to find faults, but there remains a question of how reliable it will prove to be in practice. That is, how likely is it that all equally unique test sets will find a given fault. This 'consistency' question is addressed by RQ3.

RQ4: Additional Fault Finding Ability: Can output uniqueness augment whitebox techniques?

Suppose RQs 1,2 and 3 indicate that correlations exist between output uniqueness and both whitebox coverage and consistent fault finding. This would provide evidence that we could use output uniqueness to achieve structural coverage when whitebox techniques are inapplicable. However, what if whitebox techniques *are* applicable; should we still consider using output uniqueness? RQ4 addresses this question. It asks whether output uniqueness criteria are also complementary to structural coverage criteria; do they find additional faults missed by whitebox techniques?

3.1 Experimental Design

This section describes the subjects, measures and analysis tools used in the empirical study.

3.1.1 Subjects

The web applications we used are described in Table 1. These applications range from 800 LoC to 22K LoC and have been used by other research on web testing [4, 7].

For each application, we collected all test cases generated from a tool called SWAT from previous work [4] to form a pool of test cases. This pool is used for sampling in the experiments performed for the empirical study. Test cases are collected from those generated for each of the three variations of the search based algorithms implemented for SWAT. Every test case is a sequence of one or two requests. That is, a test case consists of two requests when login is required for the action in the second request to be performed.

Details about the size and performance of each pool of test data for each application are provided in Table 2.

Table 1: The web applications used in the study.

App Name	Version	PHP Files	PHP LoC	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
PHPBB2	2.0.21	78	22,280	Customisable web forum

3.1.2 Measures

Coverage, fault detection and output uniqueness are measured and evaluated for the empirical study. For coverage: path, branch and statement coverage are measured. Statement coverage and branch coverage are widely used in research to measure and compare the effectiveness of test data generation approaches [7, 14, 27, 38] and in approaches that specifically aim to maximise such coverage [15, 26, 34, 39].

Path coverage is a stronger criterion that subsumes both branch and statement coverage. The 'all paths' criterion is infeasible, because there are usually infinitely many paths. Nevertheless, we can *measure* the number of distinct paths covered by a test suite and thereby measure path coverage.

We use an automated oracle to check for fault revelation. That is, our oracle automatically reports PHP and SQL execution errors parsed from PHP error log files and the output HTML pages of each test case. Only faults that are caused by a unique code location and have a distinct type are counted (to avoid double counting of faults). We use an automated oracle because it is unbiased and is unaffected by the experimenters' involvement in the evaluation.

Table 2 reports the size of each test pool, the total number of faults that can be detected, as well as the maximum paths, branches and statements covered by the test pool. The number of distinct outputs measured according to each output uniqueness criterion is also reported.

The pool of test data for Schoolmate is considerably larger than that for the other applications ($\sim$ 73k test cases compared to $<$ 23k). PHPSysInfo has the fewest test cases (1,130). However, these pools were not altered by reducing the Schoolmate pool or expanding the PHPSysInfo pool. We refrain from such interference with the test pools to avoid any possibility of experimenter bias.

In our experiments, each sample from the pool is the subject of a pairwise comparison of blackbox and whitebox coverage achieved. This is used in a correlation analysis. Because the measurements are paired, we are always comparing like-for-like in the inferential statistical analysis.

Although all test cases in the pool are unique, the analysis shows that, even for the strictest output uniqueness definition, only a small percentage of test cases produce unique outputs (ranging from 3-28%). The only exception is PHPSysInfo, for which nearly all outputs are unique for both the OU-All and OU-Text criteria. This is caused by the application displaying several data items that are time sensitive on the output page: Execution time and system up time.

3.1.3 Analysis Tools

Xdebug was used to record statement coverage. Xdebug cannot be used for branch and path coverage because it does not produce a trace of statements executed.

Table 2: Test data information and output analysis information: Number of test cases, faults found and total paths, branches and statements covered by the subject test data for each of the 6 web applications together with the number of distinct outputs for each of the output uniqueness definitions.

			Whitebox Coverage			Blackbox OU-						
App Name	Test Cases	Faults	Paths	Branches	Statements	All	Text	Struct	Hidden	Sub types	Seq	Last Text
FAQForge	7,233	67	176	96	516	1,287	1,175	1,049	896	78	55	7
Schoolmate	72,674	201	333	570	2,973	1,982	638	1,489	1,464	326	271	248
Webchess	9,377	98	209	460	2,097	1,608	408	1,556	1,316	151	38	15
PHPSysInfo	1,130	6	341	476	4,841	1,047	1,047	707	14	14	14	884
Timeclock	10,671	186	439	831	5,386	3,014	782	2,638	249	111	80	89
PHPBB2	22,379	79	2,030	1,337	7,807	5,617	1,602	4,138	513	92	82	27

Therefore, the applications' code was instrumented to produce execution traces that can be used to measure branch and path coverage. This instrumentation affects neither order nor frequency of execution. A test harness was developed to execute test cases and analyse the output. Statistical analysis and data visualisation was performed using R[1].

3.2 Experiments and Discussion

This section describes the experimental methodology and discusses the results obtained for each of the analyses we performed, answering each of the four research questions and describing actionable findings and threats to validity.

3.2.1 RQ1: Structural Coverage Ability

To investigate the correlation between structural coverage criteria and output uniqueness criteria, we randomly created N test suites from the test pool of each application. Each individual test suite is composed of between 10 and 500 randomly selected test cases from the pool of all test cases generated by the web testing tool SWAT.

For each test suite we measured statement, branch and path coverage as well as the number of unique outputs based on all output uniqueness criteria. We then calculated Spearman's rank correlation coefficient between each of the three whitebox coverage measures and the number of unique outputs for each of the seven output uniqueness definitions.

Spearman's rank correlation coefficient was chosen to assess correlation because it is a non-parametric statistical test and, therefore, makes assumptions about neither the distribution of faults found, nor coverage nor output uniqueness. We repeat the experiment 30 times for each application.

This gives us 30 different measurements of the Spearman rank correlation, which we will depict using box plots. These box plots give an indication of the variance in correlations observed and, thereby, an assessment of the reliability of conclusions that can be drawn about the true overall correlation (over all possible test suites).

Figure 3 illustrates the overall 'nested' structure of the evaluation experiments. In total 18,600 test suites were sampled for each application (111,600 test suites in total). This provides us with a large set of experimental data on which to base robust conclusions regarding the research questions.

We first established a suitable choice for the parameter N to the experimental approach outlined in Figure 3. This parameter determines the number of test suites used in each computation of a Spearman rank correlation. If N is insufficiently large then there would be too great a variation in correlation values observed. Such a large variance would inhibit our ability to draw firm conclusions regarding the true correlation over all possible test suites.

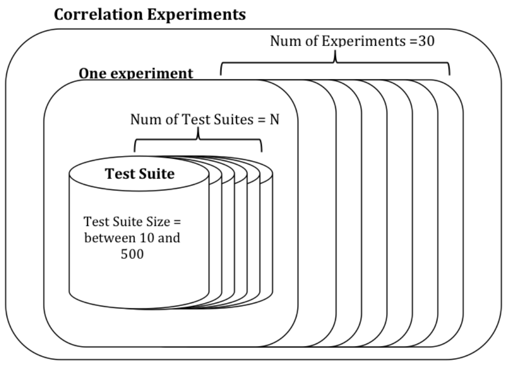


Figure 3: Framework of the experiments performed for the study. The overall framework was repeated with three values for N: 20, 100 and 500. The nested repetition required by inferential statistical analysis meant that we considered 111,600 test suites in total, over all experiments.

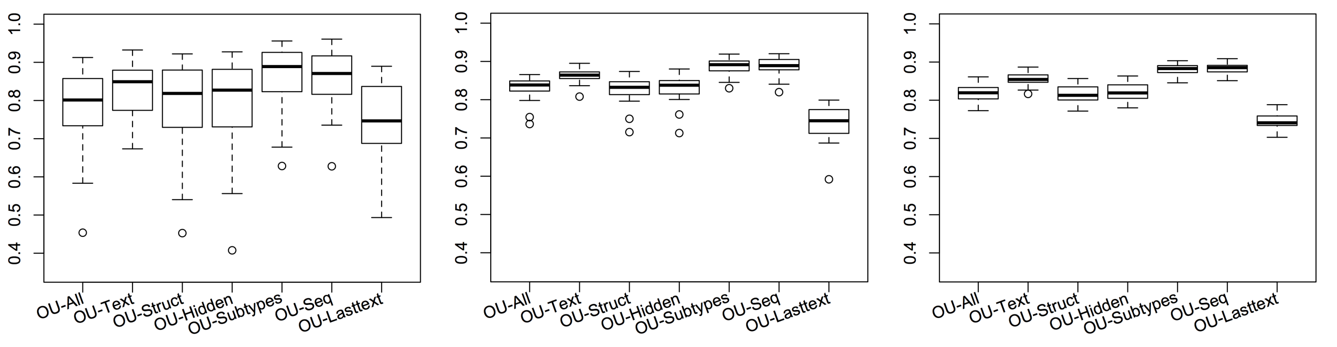
We experimented with $N = 20$, $N = 100$ and $N = 500$, examining the correlation between output uniqueness and whitebox coverage. Once the value $N = 500$ was reached it had become clear that there was little variation in correlation coefficients over all three whitebox criteria and so $N = 500$ was selected as the final choice for N.

Space does not permit us to show results for all criteria. However, Figure 4 illustrates the way in which increasing N effectively reduces the 'error bars' on the median correlation coefficient for one of the three: branch coverage. The results for statement and path coverage are similar. As can be seen from Figure 4, the variance in correlation coefficients observed is unacceptably large for $N = 20$, but it is considerably narrowed at $N = 500$.

We will use this value of N to answer RQ1 (and the subsequent RQs). Figure 5 presents the results of correlation analysis. Each of the six rows of box plots in Figure 5 relates to one of the six applications, while each of the three columns relates to one of the three whitebox criteria: statement, branch and path coverage. In each row and column of Figure 5 we find a subfigure depicting seven boxplots. These seven boxplots report the correlation between the seven output uniqueness criteria and the whitebox criterion for that column and the web application for that row.

For all six applications, the results clearly reveal a strong correlation between six of the output uniqueness criteria (all except OU-LastText) and all three whitebox coverage criteria. Path coverage exhibits the strongest correlation in general.

[1]http://www.r-project.org/



(a) 20 suites (b) 100 suites (c) 500 suites

Figure 4: Effect of test suite set size (N) on correlation coefficient variance for $N = 20, 100, 500$ over 30 different experiments for Schoolmate. Results for the other five web applications are similar.

We observed that the correlations are less strong for PHP-SysInfo for the three criteria OU-Hidden, OU-Subtypes and OU-Seq. However, recall that for these three criteria, our test pool contained few distinct outputs (only 14 in total; an order of magnitude fewer than for other criteria). For all other web applications and all of the other output uniqueness criteria for PHPSysInfo, the correlations are strong.

In summary, the answer to RQ1 is that output uniqueness criteria were strongly correlated to structural coverage.

3.2.2 RQ2: Fault Finding Ability

To investigate the correlation between fault finding ability of a test suite and structural coverage and output uniqueness criteria, Spearman's rank correlation coefficient was calculated between the number of distinct faults found and each of the output uniqueness criteria, structural coverage criteria and test suite size for each set of test suites.

Figure 6 reports the results of this analysis in six subfigures; one for each web application studied. We report the correlation between faults found and test suite size in each subfigure because we *know* size is correlated with fault finding. Therefore, correlation values for size provide a useful comparator for the other correlations observed.

The results show that for 5 of the 6 applications and all output uniqueness criteria except OU-LastText, a strong correlation exists between output uniqueness and fault finding ability. Output uniqueness is typically as strongly or more strongly correlated to fault finding than test suite size. This correlation is also at least as strong as those observed between fault finding and structural coverage.

In summary, the answer to RQ2 is that output uniqueness criteria were strongly correlated to fault finding and performed at least as well as whitebox criteria at fault finding.

3.2.3 RQ3: Fault Finding Consistency

To investigate the reliability of output uniqueness criteria, we analyse the consistency of fault finding for each output uniqueness definition. If all test cases that produce the same output always find the same faults, these faults are very likely to be detected by any test suite that satisfies the output uniqueness criteria. The analysis is also applied to structural coverage to facilitate comparison.

Table 3 reports the results of the analysis. The last five columns of the figure can be ignored for now, since these address RQ4 and will be discussed in the next section.

The first column is the web application for which results are reported in the remaining columns. The second column reports the total number of faults found over all techniques. The third and fourth columns report the number of faults found consistently and inconsistently by branch coverage, according to our definitions of consistency explained below.

A fault is **found consistently by branch coverage** if there exists a branch where all tests that cover the branch reveal the fault, and it is found inconsistently, if no such branch exists. These results from the third and fourth column (branch coverage consistency) can be compared with the corresponding results for each of the seven output uniqueness criteria. The sixth and seventh columns report the numbers of faults found consistently and inconsistently by our output uniqueness criteria.

A fault is **found consistently by an output uniqueness** criterion OU, if there exists an output o that is unique according to OU and all test cases that produce o reveal the fault. It is found inconsistently if it is found by some OU test suites but no such o exists.

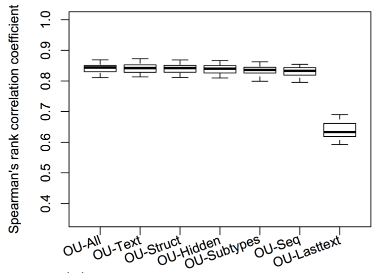
The results show that, for all applications except PHPBB2, at least one output uniqueness criterion finds more faults consistently than branch coverage. Indeed, for all applications, test suites that satisfy OU-All, OU-Text and OU-Struct criteria find more faults consistently than branch coverage. The number of faults found consistently by output uniqueness decreases as the criteria become less strict.

These results confirm that the more strict criteria can be effective as test adequacy criteria when structural coverage cannot be measured. Indeed, OU-All consistently finds 92% of the faults consistently found by branch coverage.

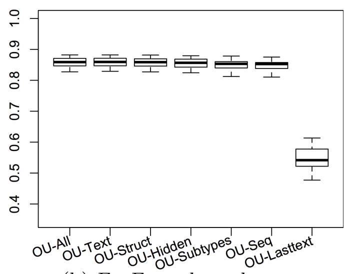
In summary, the answer to RQ3 is that for several output uniqueness criteria, fault finding ability was at least as consistent as it was for whitebox criteria.

3.2.4 RQ4: Additional Fault Finding Ability

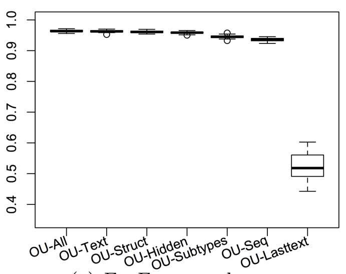
The last five columns of Table 3 address the complementarity of output uniqueness. The columns headed '$\cup$' and '$\cap$' show, respectively, the union and intersection of faults found consistently by branch coverage and faults found consistently by output uniqueness. The column headed '$B - O$' reports faults found consistently by branch coverage but not by output uniqueness, while the column headed '$O - B$' reports faults found consistently by output uniqueness but not branch coverage. Finally, the column headed '$F - (\cup)$' shows the faults that are found by either technique, but which are not found consistently by either.



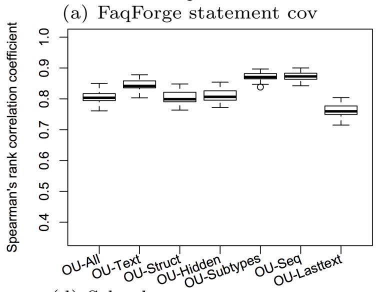
(a) FaqForge statement cov



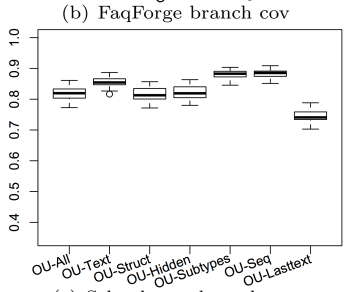
(b) FaqForge branch cov



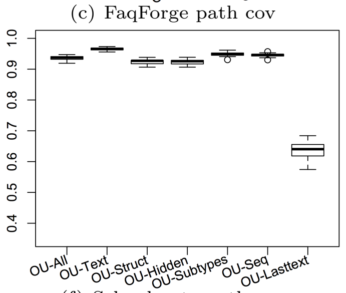
(c) FaqForge path cov



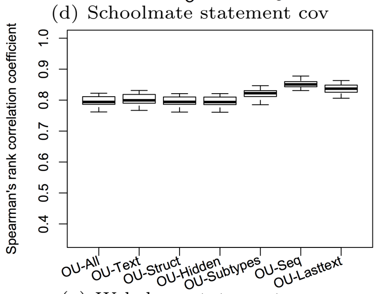
(d) Schoolmate statement cov



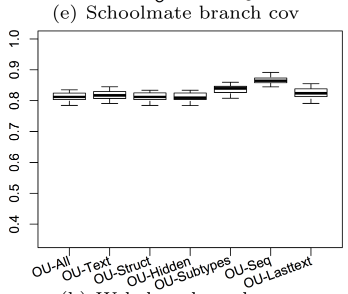
(e) Schoolmate branch cov



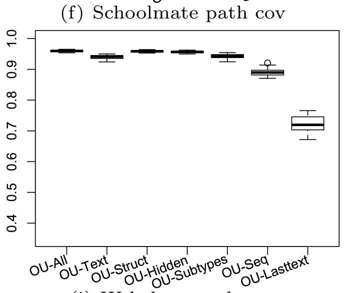
(f) Schoolmate path cov



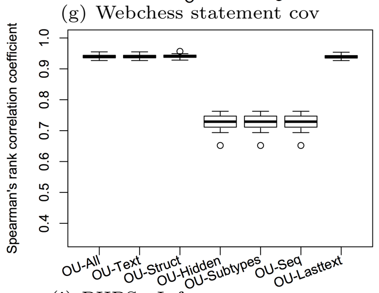
(g) Webchess statement cov



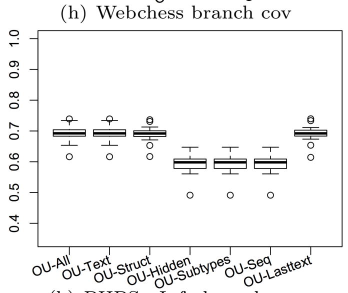
(h) Webchess branch cov



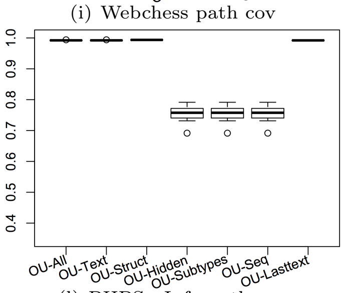
(i) Webchess path cov



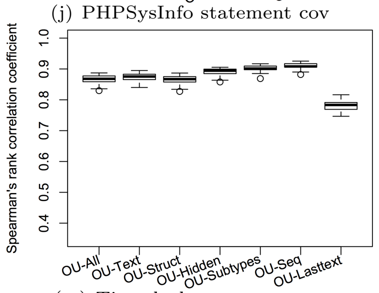
(j) PHPSysInfo statement cov



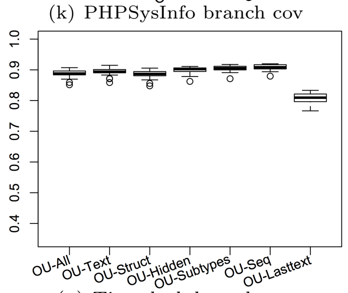
(k) PHPSysInfo branch cov



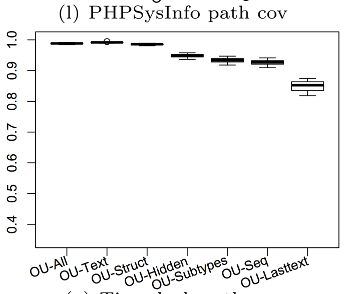
(l) PHPSysInfo path cov



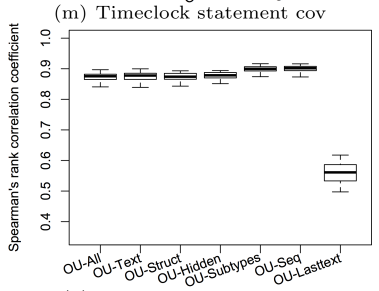
(m) Timeclock statement cov



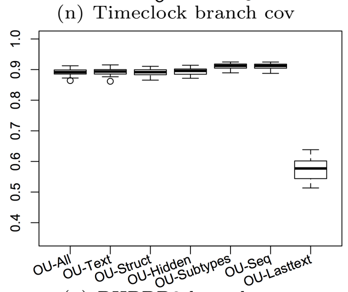
(n) Timeclock branch cov



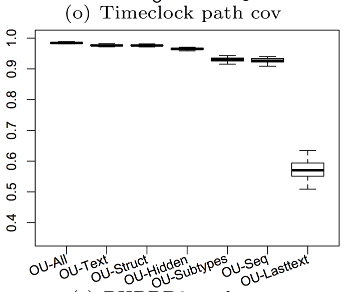
(o) Timeclock path cov



(p) PHPBB2 statement cov

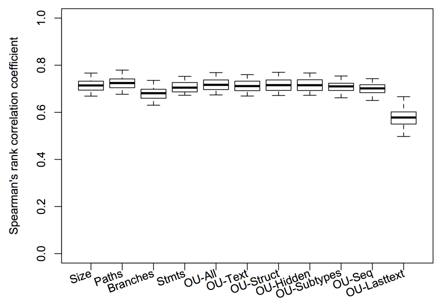


(q) PHPBB2 branch cov

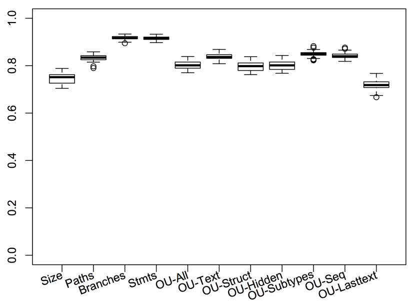


(r) PHPBB2 path cov

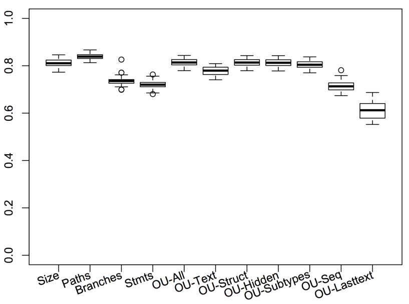
Figure 5: Variations in Spearman's rank correlation coefficient between structural coverage and output uniqueness for 500 test suites of a random size between 10 and 500 test cases over 30 different experiments.



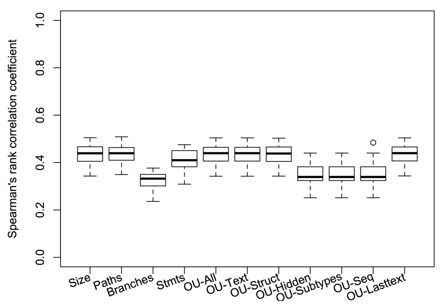
(a) FaqForge



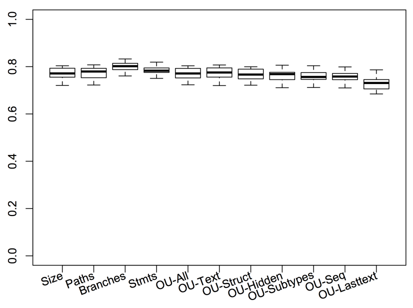
(b) Schoolmate



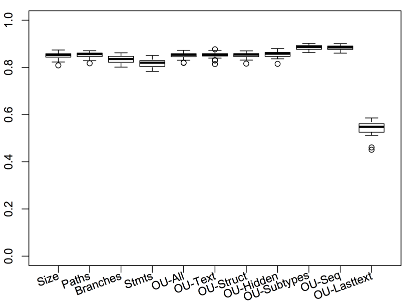
(c) Webchess



(d) PHPSysinfo



(e) Timeclock



(f) PHPBB2

Figure 6: Variations in Spearman's rank correlation coefficient between fault finding and structural and output criteria for 500 test suites of a random size between 10 and 500 test cases over 30 different experiments for the 6 applications. Output uniqueness criteria exhibit strong correlation to fault finding, comparing favourably to both the correlations for size of test suite and those for the three whitebox coverage criteria.

The results provide evidence to support the claim that output uniqueness and branch coverage are complementary. We also investigated complementarity for both statement and path coverage. Though space does not permit us to present the results here in the same level of detail as for branch coverage[2], we can report that results for statement coverage are similar to those for branch coverage. We can also report that, even for path coverage, for 4 of the 6 applications, the faults found consistently by output uniqueness criteria are complementary to those found by path coverage.

Perhaps, most important for the impact of output uniqueness, we observe, over all programs studied, that our OU-All blackbox text suites consistently found 47% of the additional faults not found by branch coverage.

In summary, the answer to RQ4 is that output uniqueness proved to be complementary to whitebox criteria; it found many faults that whitebox techniques left undetected.

3.3 Future Work and Actionable Findings

In this section we briefly describe directions for future research and actionable findings. The evidence we present in this paper derives from experiments with six real-world web applications; we have not presented evidence concerning non-web-based applications. Furthermore, some of our output uniqueness criteria are specifically defined for the web, drawing inspiration from the structure inherent in HTML output. Results for these criteria are clearly not applicable to other (non-web-based) testing scenarios. A natural next step would be to investigate whether similarly strong correlations to coverage and fault detection are enjoyed by output uniqueness with non-web-based applications.

Some of our output uniqueness criteria are specifically web-orientated. However, we also observed that our purely text-based output uniqueness criterion, OU-Text, enjoyed a high correlation with whitebox criteria. Though our empirical results are currently confined to web applications, OU-Text is, in principle, applicable to any application that produces text output. This suggests future work to investigate whether OU-Text (and related, more general, output uniqueness criteria) might be useful in non-web-based application testing scenarios.

Future work will also develop new blackbox test generation techniques that aim to maximise output diversity. The test cases generated by such new techniques may be expected, based on our findings, to achieve high whitebox coverage and high fault finding effectiveness.

Developing such techniques is not trivial; one possible approach is to dynamically learn the effect of changing each input (or combination of inputs) on the output to focus test generation efforts on inputs that, when changed, yield interesting outputs. Progress or 'coverage' of the produced test cases could be measured by estimating the number of possible outputs using statistical estimation techniques.

Output uniqueness may also have a contribution to make by reducing the impact of the Oracle Problem [21, 22, 40]. Where the human tester has to play the role of oracle, output uniqueness will help in two ways: it can reduce the number of test cases to be considered through selection and prioritisation, and it may ensure that those tests that the human does consider will have highly distinct outputs, potentially making them easier to check manually.

We believe our findings also include several actionable findings for testing practitioners. Blackbox testing techniques are common in software testing practice [8, 9].

[2]Full results available in PhD thesis[3]

Table 3: Consistency and complementarity of fault finding ability. For each application we report: All faults (F), faults found by branch coverage (B), faults found by output uniqueness (O), consistently found (C), inconsistently found (I) and those found by either technique (∪) and by both (∩)

		B			O				$B-$	$O-$	$F-$
App Name	F	C	I	OU-	C	I	∪	∩	O	B	(∪)
				All	55	12	58	34	3	21	9
				Text	51	16	57	31	6	20	10
				Struct	55	12	58	34	3	21	9
FAQForge	67	37	30	Hidden	52	15	55	34	3	18	12
				Subtypes	48	19	54	31	6	17	13
				Seq	48	19	54	31	6	17	13
				LastText	13	54	37	13	24	0	30
				All	162	39	176	88	14	74	25
				Text	134	67	154	82	20	52	47
				Struct	148	53	168	82	20	66	33
Schoolmate	201	102	99	Hidden	139	62	163	78	24	61	38
				Subtypes	105	96	145	62	40	43	56
				Seq	84	117	135	51	51	33	66
				LastText	14	187	103	13	89	1	98
				All	77	21	77	44	0	33	21
				Text	56	42	58	42	2	14	40
				Struct	71	27	77	38	6	33	21
Webchess	98	44	54	Hidden	64	34	75	33	11	31	23
				Subtypes	59	39	70	33	11	26	28
				Seq	17	81	46	15	29	2	52
				LastText	19	79	45	18	26	1	53
				All	6	0	6	4	0	2	0
				Text	6	0	6	4	0	2	0
				Struct	5	1	6	3	1	2	0
PHPSysInfo	6	4	2	Hidden	3	3	4	3	1	0	2
				Subtypes	3	3	4	3	1	0	2
				Seq	3	3	4	3	1	0	2
				LastText	2	4	6	0	4	2	0
				All	139	47	163	112	24	27	23
				Text	125	61	149	112	24	13	37
				Struct	90	96	163	63	73	27	23
Timeclock	186	136	50	Hidden	55	131	139	52	84	3	47
				Subtypes	51	135	138	49	87	2	48
				Seq	50	136	138	48	88	2	48
				LastText	27	159	136	27	109	0	50
				All	70	9	74	66	4	4	5
				Text	55	24	74	51	19	4	5
				Struct	61	18	74	57	13	4	5
PHPBB2	79	74	9	Hidden	48	31	74	44	26	4	5
				Subtypes	45	34	74	41	29	4	5
				Seq	43	36	74	39	31	4	5
				LastText	12	67	71	11	59	1	8

Our findings offer the potential for practitioners to further leverage existing blackbox techniques and tools to achieve whitebox testing criteria. A web application tester could generate pools of test cases using existing blackbox tools and techniques and subsequently select subsets guided by the output uniqueness criteria. Our findings also indicate that subsets so-selected will find additional faults not found by whitebox techniques.

Our findings can be used to improve the efficiency of existing test practices. Output uniqueness can be used to guide the necessary selection and prioritisation processes [42] that arise when the practice of executing all available test cases is infeasible due to the large number of test cases. Di Nardo et al. [12] suggested that novel prioritisation/selection approaches are required since relying on coverage alone is unlikely to provide significant improvements.

3.4 Threats to Validity

Internal threats: Internal threats to validity are factors that affect the dependent variables and are not controlled in the experiments. The test suites selected were all generated randomly in the same manner. The choice of test suite size might have an effect on results however a random size between 10 and 500 was chosen, because test suites smaller than 10 might not display a diversity in coverage and number of distinct outputs and 500 is larger than the largest test suite size that was originally produced by the testing tool.
External threats: External threats to validity are threats that limit the ability to generalise results. There are three main threats to external validity: the choice of applications studied, the fault oracle used and the source of test cases in the test pools.

We studied six applications. An empirical study of more applications is needed before being able to generalise results. However, the selected applications represent different domains and are used by real users. They also have diverse sizes and architectures.

The faults measured to assess effectiveness are faults that can be detected automatically. To generalise results to other types of fault, an investigation of how these automatically detectable faults relate to other types of faults is needed or some other oracle must be found. However, the faults reported and used in this study are real faults, not seeded faults, and they are checked using a fully automated oracle, freeing the study from experimenter bias in the selection of faults studied.
Construct threats: Construct threats are related to the measures used in the experiments and their ability to capture what they are measuring. Fault finding ability was selected to measure the effectiveness of test suites and the different criteria as it is the aim of any testing process. Path, branch and statement coverage were selected to represent structural coverage because branch and statement coverage are widely used in the industry and research while path coverage is the strongest structural coverage criteria and thereby subsumes other structural and dataflow criteria not measured in the study.

4. RELATED WORK

The concept of output uniqueness draws on a rich intellectual history in the development of software testing that can be traced back to the test selection criteria proposed by Goodenough and Gerhart [20], and refined by Weyuker and Ostrand [41] and Richardson and Clarke [36]. Output uniqueness is closer, conceptually, to the idea of revealing subdomains introduced by Weyuker and Ostrand, because it is entirely a blackbox approach, whereas the equivalence partitioning method of Richardson and Clarke involves both black and whitebox techniques. Ostrand and Balcer [30] also developed the concept of revealing subdomains by introducing the category partition method, in which a specification is analysed in order to partition the input space into categories.

All of these approaches share the common motivation that the input space can be partitioned into equivalence classes, each of which capture a sub-computation. For Richardson and Clark, an equivalence class is an undecidable (but practically approximatable) combination of the whitebox path domain and the blackbox specification domain.

For Ostrand, Weyuker and Balcer, an equivalence class s has the purely blackbox, yet also undecidable, property that an element x of s leads to a correct output iff all elements of s lead to correct output. Though unachievable in practice, this has remained theoretically interesting and appealing, because it reduces program verification to a potentially enumerable set of tests; an idea that has resonated in work that combines verification and testing [17, 23, 24].

Output uniqueness shares a similar motivation: to consider tests which, in some sense, yield different outputs as being preferable to those that yield, in some sense, the same output. However, previous techniques focused on the *input* space, seeking input sets for which computation is *similar* according to a *semantic* abstraction. Output uniqueness focuses on the *output* space, seeking outputs that are *different* according to some *syntactic* abstraction. In this paper, we introduced seven such syntactic abstractions (pertinent to web applications), though many other abstractions (and for many other systems) are clearly possible and could be explored in future work.

Our focus on syntactic output differences also makes the underlying decision procedure decidable, though there may be no upper bound on the number of distinct outputs possible for some systems. Unlike other test adequacy criteria, such as branch coverage, which have an upper bound for all programs (though whether maximum coverage has been achieved remains undecidable), a program with conceptually infinitely many outputs, has no upper bound on output uniqueness.

Many studies [6, 16, 25, 29] have investigated the effectiveness of structural and dataflow test adequacy criteria. Such studies investigate whether these criteria independently influence fault finding ability or whether any increased effectiveness is a side effect of the larger test suite sizes required to satisfy the criteria. Previous studies have provided some evidence that structural and dataflow coverage does influence fault detection effectiveness. However, results have also showed that those criteria are not the only factor that affects fault detection. In this paper we provide further evidence supporting the findings of these studies, proposing and reporting on a new family of blackbox criteria that are based on output and are complementary to whitebox coverage in finding faults.

Dynamic Symbolic Execution [7, 19, 38] and Search Based approaches [2, 28] are two widely studied whitebox testing approaches that have been applied to web applications [4, 7]. Both approaches aim to maximise structural coverage of the application under test. Structural coverage criteria have been also used in test selection, prioritisation and minimisation [42]. The output uniqueness criteria presented in this paper are proposed as a surrogate for these approaches when whitebox approaches are not applicable.

Web application output has been used to understand the application's interface [13], automate the oracle [37] or help crawlers in automatically completing online `Forms` [35]. This previous work has not studied output uniqueness. For example, Elbaum et al. [13], in their interface identification approach, examined the output to infer properties about the inputs of the application to understand relationships between input fields. Our work is concerned with analysis of output to maximise differences (output uniqueness). The two approaches might be combined in future work to design an output uniqueness test generation approach.

In feedback directed random testing (RANDOOP) [31, 32, 33], the feedback from executing test sequences, such as exceptions and violations, is used to exclude sequences that cause errors from the generation process. We propose to use the feedback from executing test cases to diversify output, excluding test cases that produce already seen output in favour of those that maximise 'output novelty' (uniqueness).

Yuan and Memon [43, 44] observed the effect of executing test sequences on the state to identify interacting states as a means of generating tests for Graphical User Interface (GUI) testing. These states are then used to generate new sequences that are effective in finding new faults. Their approach focuses on test sequences and their interactions while we propose to diversify the output of the entire test suite.

Adaptive Random Testing (ART) [10, 11] aims to diversify the inputs in test cases while our approach focuses on diversifying the output. In our empirical study, all test cases were unique, i.e., the combination of input values in each test case does not appear in any other test case used. However, we did not calculate any similarity measures between input values as is suggested by ART. Therefore, the two approaches might be complementary and further investigations are needed to establish the relationship between these two approaches.

Many random test data generation tools are available: Web crawling tools for web applications are available both commercially and as research prototypes [18]. These tools are able to generate a large number of test cases. Our proposed OU criteria can be used as a post-processing step to select a subset of the test cases generated by these tools. According to the findings we present in this paper, it would be reasonable to hope that the test cases thus selected will tend to favour structural coverage and fault finding ability.

Our previous NIER paper [5] was the first paper to propose the idea of output uniqueness, introducing four output uniqueness criteria. In this paper we extend these four with three additional criteria and report results of an empirical study that demonstrate that output uniqueness can be used as both surrogate for whitebox testing and also as a consistent fault finding supplement.

5. CONCLUSION

In this paper we have demonstrated that the new blackbox testing criterion, output uniqueness, can be used to achieve high levels of structural (whitebox) test coverage for web applications. Specifically, we present evidence from an empirical study on six real-world web applications that indicates high levels of correlation between output uniqueness and statement, branch and path coverage.

We have also presented empirical evidence concerning the faults found by output uniqueness. The findings from this study allow us to conclude that output uniqueness is a promising new blackbox testing criterion for web applications. Not only can it be used as an effective surrogate for whitebox testing when no source code is available, it also complements whitebox testing; finding faults that whitebox testing leaves undetected.

Finally, we have presented evidence concerning the consistency of test cases selected to achieve output uniqueness. These results indicate that test suites selected to achieve output uniqueness are more consistent in the faults they find when compared to those selected for existing whitebox criteria.

6. REFERENCES

[1] W. Afzal, R. Torkar, and R. Feldt. A systematic review of search-based testing for non-functional system properties. *Inf. Softw. Technol.*, 51(6):957–976, June 2009.

[2] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege. A systematic review of the application and empirical investigation of search-based test case generation. *TSE*, 36:742–762, November 2010.

[3] N. Alshahwan. *Utilizing Output in Web Application Server-Side Testing*. PhD thesis, UCL (University College London), 2012.

[4] N. Alshahwan and M. Harman. Automated web application testing using search based software engineering. In *ASE '11*, pages 3–12, 2011.

[5] N. Alshahwan and M. Harman. Augmenting test suites effectiveness by increasing output diversity (NIER track). In *ICSE '12*, pages 1345–1348, 2012.

[6] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin. Using mutation analysis for assessing and comparing testing coverage criteria. *IEEE TSE*, 32(8):608–624, August 2006.

[7] S. Artzi, A. Kiezun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D. Ernst. Finding bugs in web applications using dynamic test generation and explicit-state model checking. *IEEE TSE*, 36:474–494, July 2010.

[8] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, 1990.

[9] A. Bertolino. Software testing research: Achievements, challenges, dreams. In L. Briand and A. Wolf, editors, *FOSE*, 2007.

[10] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse. Adaptive random testing: The art of test case diversity. *Journal of Systems and Software*, 83(1):60–66, January 2010.

[11] T. Y. Chen, H. Leung, and I. K. Mak. Adaptive random testing. In *ASIAN '04*, pages 320–329, 2004.

[12] D. Di Nardo, N. Alshahwan, L. Briand, and Y. Labiche. Coverage-based test case prioritisation: An industrial case study. In *Proceedings of the 6th International Conference on Software Testing, Verification, and Validation (ICST '13)*, pages 151–160, 2013.

[13] S. Elbaum, K.-R. Chilakamarri, M. F. II, and G. Rothermel. Web application characterization through directed requests. In *WODA'06*, pages 49–56, 2006.

[14] R. Ferguson and B. Korel. The chaining approach for software test data generation. *ACM TOSEM*, 5(1):63–86, January 1996.

[15] R. Ferguson and B. Korel. The chaining approach for software test data generation. *ACM TOSEM*, 5(1):63–86, Jan. 1996.

[16] P. G. Frankl and S. N. Weiss. An experimental comparison of the effectiveness of branch testing and data flow testing. *IEEE TSE*, 19(8):774–787, August 1993.

[17] M.-C. Gaudel. Testing can be formal, too. *Lecture Notes in Computer Science*, 915:82–96, 1995.

[18] C. Girardi, F. Ricca, and P. Tonella. Web crawlers compared. *International Journal of Web Information Systems*, 2(2):85–94, 2006.

[19] P. Godefroid, N. Klarlund, and K. Sen. DART: directed automated random testing. In *PLDI '05*, pages 213–223, 2005.

[20] J. B. Goodenough and S. L. Gerhart. Toward a theory of test data selection. *IEEE Transactions on Software Engineering*, 1(2):156–173, June 1975.

[21] M. Harman, S. G. Kim, K. Lakhotia, P. McMinn, and S. Yoo. Optimizing for the number of tests generated in search based test data generation with an application to the oracle cost problem. In *SBST'10*, 2010.

[22] M. Harman, P. McMinn, M. Shahbaz, and S. Yoo. A comprehensive survey of trends in oracles for software testing. Technical Report Research Memoranda CS-13-01, Department of Computer Science, University of Sheffield, 2013.

[23] R. Hierons, K. Bogdanov, J. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Luettgen, T. Simons, S. Vilkomir, M. Woodward, and H. Zedan. Using formal methods to support testing. *ACM Computing Surveys*, 41(2), Feb. 2009. Article 9.

[24] M. Holcombe. What are X-machines? *Formal Asp. Comput*, 12(6):418–422, 2000.

[25] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *ICSE'94*, pages 191–200, 1994.

[26] S. Khor and P. Grogono. Using a genetic algorithm and formal concept analysis to generate branch coverage test data automatically. In *ASE '04*, pages 346–349, 2004.

[27] K. Lakhotia, P. McMinn, and M. Harman. An empirical investigation into branch coverage for C programs using CUTE and AUSTIN. *Journal of Systems and Software*, 83:2379–2391, December 2010.

[28] P. McMinn. Search-based software test data generation: a survey. *STVR*, 14(2):105–156, June 2004.

[29] A. S. Namin and J. H. Andrews. The influence of size and coverage on test suite effectiveness. In *ISSTA '09*, pages 57–68, 2009.

[30] T. J. Ostrand and M. J. Balcer. The category-partition method for specifying and generating functional tests. *Communications of the ACM*, 31(6):676–686, June 1988.

[31] C. Pacheco and M. D. Ernst. Randoop: feedback-directed random testing for java. In *OOPSLA '07*, pages 815–816, 2007.

[32] C. Pacheco, S. K. Lahiri, and T. Ball. Finding errors in .NET with feedback-directed random testing. In *ISSTA '08*, pages 87–96, 2008.

[33] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball. Feedback-directed random test generation. In *ICSE '07*, pages 75–84, 2007.

[34] S. Park, B. M. M. Hossain, I. Hussain, C. Csallner, M. Grechanik, K. Taneja, C. Fu, and Q. Xie. Carfast: achieving higher statement coverage faster. In *FSE '12*, pages 35:1–35:11, 2012.

[35] S. Raghavan and H. Garcia-Molina. Crawling the

hidden web. In *VLDB '01*, pages 129–138, 2001.

[36] D. J. Richardson and L. A. Clarke. A partition analysis method to increase program reliability. In *Proceedings of the 5th International Conference on Software Engineering*, pages 244–253. IEEE Computer Society Press, Mar. 1981.

[37] S. Sampath, R. C. Bryce, G. Viswanath, V. Kandimalla, and A. G. Koru. Prioritizing user-session-based test cases for web applications testing. In *ICST '08*, pages 141–150, 2008.

[38] K. Sen, D. Marinov, and G. Agha. CUTE: a concolic unit testing engine for C. In *ESEC/FSE-13 '05*, pages 263–272, 2005.

[39] M. L. Soffa, A. P. Mathur, and N. Gupta. Generating test data for branch coverage. In *ASE '00*, pages 219–227, 2000.

[40] E. J. Weyuker. On testing non-testable programs. *The Computer Journal*, 25(4):465–470, Nov. 1982.

[41] E. J. Weyuker and T. J. Ostrand. Theories of program testing and the the application of revealing subdomains. *IEEE Transactions on Software Engineering*, 6(3):236–246, May 1980.

[42] S. Yoo and M. Harman. Regression testing minimisation, selection and prioritisation: A survey. *Journal of Software Testing, Verification and Reliability*, 22(2):67–120, 2012.

[43] X. Yuan and A. M. Memon. Using GUI run-time state as feedback to generate test cases. In *ICSE '07*, pages 396–405, 2007.

[44] X. Yuan and A. M. Memon. Generating event sequence-based test cases using GUI runtime state feedback. *IEEE TSE*, 36(1):81–95, January 2010.