2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)

Measuring Combinatorial Coverage at Adobe

Riley Smith, Darryl Jarman, Jared Bellows
Adobe Systems Inc.
Utah, USA
{rilsmith, djarman, jbellows}@adobe.com

Richard Kuhn, Raghu Kacker
NIST
Maryland, USA
{d.kuhn, raghu.kacker}@nist.gov

Dimitris Simos
SBA Research
Vienna, Austria
dsimos@sba-research.org

***Abstract*- Adobe offers an analytics product as part of the Marketing Cloud software with which customers can track many details about users across various digital platforms. For the most part, customers define the amount and type of data to track. In addition, customers can specify many feature combinations when reporting on this data. These features create high dimensionality that makes validation challenging for some of the most critical components of the Adobe Analytics product. One of these critical components is the reporting engine. This component has a validation framework often qualitatively considered within the engineering organization as highly effective. However, the effectiveness of this framework has never been quantitatively measured. Due to recent applications of combinatorial testing, the Analytics Tools team determined to use combinatorial coverage measurements (CCM) to evaluate the effectiveness of the Replay validation framework. In this paper, we therefore report the practical application of combinatorial coverage measurements to evaluate the effectiveness of the validation framework for the Adobe Analytics reporting engine. The results of this evaluation show that combinatorial coverage measurements are an effective way to supplement existing validation for several purposes. In addition, we report details of the approach used to parse moderately nested data for use with the combinatorial coverage measurement tools.**

Keywords- Combinatorial Testing, Combinatorial Coverage Measurement, Industry, Application

I. INTRODUCTION

Originating from web analytics, the Adobe Analytics product has evolved into a customer marketing platform allowing users to instrument data collection across many digital platforms for real-time reporting. Users of Adobe Analytics configure the amount and type of data to track in addition to the type and complexity of reports. The available configurations result in high dimensionality for any elements of the system that interact with the collected data. For example, the reporting engine accesses the stored data and calculates user-defined metrics from the contents. As the product has evolved, the number of configurable elements has only increased for these components.

Given this domain knowledge, traditional validation of the reporting engine has relied on reusing actual customer requests. This provides the benefit of exercising the parts of the system commonly used by users. This approach was generally seen as a practical solution to exercise the input space based on the assumption that the input space was too broad to systematically cover. Over time, detected faults exposed interactions not covered by the existing approach. These detected faults show that existing validation fails to provide comprehensive coverage of the system.

A key observation of combinatorial testing maintains that software faults are generally caused by the interactions between a limited (small) number of input parameters [1]. Generally, a t-way combinatorial test covers all t-way interactions. After discovering combinatorial testing, initial investigations revealed several reports showing the effectiveness in practical industry applications [2]. Despite many being labeled as "uncontrolled" applications and studies [3], these reports prompted the internal tools team within Adobe Analytics to quantitatively evaluate the coverage of the traditional validation.

In this paper, we consequently report an industry application of combinatorial coverage measurement (CCM) to the Adobe Analytics reporting engine. The analysis is considered successful if the measurements produce the desired result: quantitatively exposing the lack of coverage to prompt improvement in existing validation. The results of the combinatorial measurements indeed suggest a gap in coverage that could be addressed with combinatorial testing.

It is important to note that the subject system is large in terms of both lines of code and number of input parameters. In addition, the input parameters have complex constraints. This has three main implications: (1) the input space was minimized to make the analysis both more manageable and precise and (2) we were only able to use existing tools to generate 2-way and 3-way measurements for most aspects of this analysis. Consequently, we also report the details of the approach used to minimize the input space. The approach consists of four main steps: (1) collect test cases from previous validation, (2) parse the previous test cases, (3) normalize the values for the input parameters, and (4) format the parsed test cases for use with Automated Combinatorial Testing for Software (ACTS) [4] and Combinatorial Coverage Measurement Command-line (CCMCL) [5] tools provided by the National Institute of Standards and Technology (NIST).

The remainder of the paper proceeds as follows. In section II, we provide background for the subject system. Section III reports the approach and setup of the application including the input space modeling process. Section IV details the results of this application.

978-1-7281-0888-9/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSTW.2019.00052

II. Background

A. General Workflow

For the Analytics product, Adobe provides customers with a software development kit (SDK). Customers use the SDK to instrument web sites and mobile applications. The instrumented applications then send data to the Adobe Analytics data collection pipeline based on this implementation. This collection process supports thousands of individual data fields. Eventually, this pipeline converts the data into a columnar format where it waits to meet certain conditions. Once meeting the conditions, the collection system exports the data to the compression algorithm. This algorithm transforms the data to another format for long term storage. The processing system reads the compressed files and transforms the data for reporting. Users generate report requests from the data in these newly created files. These reports relate to general metrics used to evaluate the success of digital marketing platforms (i.e. website, mobile application) to produce the desired result (i.e. click, purchase, etc). For example, a conversion report might return ten percent if one person out of ten met a defined condition like purchasing a particular product.

B. Reporting Engine Details

When generating report requests, users can select many different report types and options to change report behavior. Table I and Table II summarize these most common report elements. Each of these report elements can be used with any of the thousands of potentially collected data fields mentioned above.

TABLE I. Adobe Analytics Report Types

Report Type	***Description***
Ranked	Report field ranked by metrics related to each value of field
Conversion/Average (CA)	Returns conversion or average rate for users of the target application

TABLE II. Adobe Analytics Report Options

Report Option	***Description***
Aggregation	Aggregates data from user-defined rules
Breakdown	Expands report by given field
Classification	Dynamically labels data by pattern
Filter	Filters data based on user-defined criteria for given field
Metric	Defines metrics to calculate in context of selected fields

C. Existing Validation

The engineers of the reporting engine currently have two main validation strategies: (1) regression testing and (2) integration testing. The existing regression testing involves randomly selecting recent customer requests for reuse. Meanwhile, the existing integration testing stores specific customer requests that have historically proved problematic. The validation framework executes the selected report requests with the current live code and the new proposed code. The framework then compares the output. Generally, engineers execute the framework once before delivering new code versions. However, the execution only occurs for an arbitrary period. This usually results in about 40,000 executed test cases. Despite flaws, this approach has proved both effective and practical. However, the current validation lacks any quantification of coverage or stopping criteria.

This background information shows how the reporting system depends on two different sets of input parameters: (1) the data collected by the customer and (2) the details of report requests. A separate paper [6] focuses on applying combinatorial testing to the former while this paper focuses on the latter. Furthermore, this paper focuses only on the most common report types and options. For example, ranked and conversion/average reports account for 98 percent of customer reports and 100 percent of test cases completed by the existing validation during the last 6 months.

III. Approach and Setup

A. Data Preparation

Internally, the subject system uses extensible markup language (XML) to represent the elements of report requests. Usually these XML document structures consist of a root element with one or more child elements for each report option. These child elements contain additional elements that define the individual objects of that report option (Figure 1). Taking Figure 1 as an example, a customer might request a report that ranks the pages by number of page views, but only include certain pages from the site. Figure 2 shows what this might resemble. It is important to note that this is a trivial example.

```
< root attribute_1 … attribute_n >
    < child_1 attribute_1 … attribute_n >
        <nested_1 attribute_1 … attribute_n >
        …
        <nested_n attribute_1 … attribute_n >
    …
    < child_n attribute_1 … attribute_n >
< /root >
```

Figure 1

```
< ranked-report locale="en_US" field="page" >
   < filters >
     <filter value="http://icst2019.xjtu.edu.cn/index.htm" / >
     <filter value="iwct2019.sba-research.org/cfp.html" / >
   </ filters >
   < metrics >
     <metric identity="metrics/pageviews" />
   < /metrics >
 < /ranked-report >
```

Figure 2

To measure combinatorial coverage of this existing test suite as quickly and easily as possible, the data in this nested XML structure required parsing the XML into the formats supported by existing tools. For example, the CCMCL tool provided by the National Institute of Standards and Technology (NIST) accepts a comma-separated value (CSV) file. We consequently considered two approaches for appropriately organizing each nested XML document into rows.

The first option was the most obvious: flattening the nested XML such that each document becomes a single row where each tag and attribute becomes a column. For example, a row for the document in Figure 1 would have a column for the root tag, each root attribute, each child tag and attributes, and each nested child tag and attributes. This presents an easy solution while preserving context crucial to the analysis. However, this approach also greatly increases the dimensionality of the analysis by making each row match the aggregated dimensionality of the XML documents with the most complex objects for each report option. Furthermore, the dimensionality of this approach often proves unnecessary as the resulting combined row contains data not relevant to every XML document.

The second option was less obvious, but perhaps more intuitive. Instead of considering each document as an individual row, the various elements of an individual document were divided by functionality. For instance, the report type tag and attributes hold significance, but the report type holds little bearing on the functionality of the individual report options. Similarly, the interactions for an individual report option hold significance, but one report option holds little bearing on the functionality of other report options. Consequently, relevant tags and attributes from the child elements can be included with those in the root or nested elements. Taking Figure 1 as an example, the tag and attributes of the root element would be written to a CSV for the report type with additional columns to mark the existence or absence of a child element. Meanwhile, the tag and attributes of the nested elements would be written to a CSV for the report option with additional columns to capture the specific details of the child element. While losing the exact details of some context information as a result of this forced isolation, this approach preserves the essence of the context (i.e. the existence of an input parameter) for the root element. Meanwhile, the nested elements maintain the exact details for isolated analysis of the discrete functional group (i.e. report option).

With the data organized to accommodate easy transformation and analysis, we proceed with the input space modeling of the data objects we created.

B. *Input Space Modeling*

The CCMCL tool offers two options for modeling the input space. This analysis reports the results of both options: (1) infer the input space from the CSV containing the test cases as rows or (2) use an ACTS configuration file as an input space definition. Using the first option, the CCMCL tool will collect variable values from the input file which assumes the test cases represent all values for all input parameters. That said, the combinatorial coverage measurement of the inferred input space still provides valuable insight. However, using an ACTS configuration file will provide a more meaningful view of the existing test suite considering the nuances of the existing validation. Otherwise, we assume that the random selection of recent customer requests contains a representative set of all input parameter values.

Unfortunately, the subject system does not have an input space model for easy conversion to the ACTS file format. To compensate for this, we parsed nearly 28 million XML documents from a short period of customer requests. From the nearly 28 million XML documents, we removed duplicates to reduce the parsing time. In addition, we identified another area to reduce the data set while increasing usefulness. As mentioned, the Adobe Analytics product allows customers to collect diverse data and perform various operations on that data. However, many of these fields and operations have the same form and function. Like removing duplicate XML documents above, we normalized the input parameter values to accommodate this domain knowledge by replacing references to equivalence classes with a common placeholder. This applies to many of the report options in Table II. These actions resulted in about 8 million unique XML documents. Compared to the input space inferred from the test cases, these 8 million XML documents did provide a more complete representation of the input space.

IV. Application

We use the generated CSV and ACTS files with the combinatorial coverage measurement command line tool (CCMCL) available from NIST. The tool provides a few measurements. We focused on the 2-way and 3-way coverage percentage as the best indicator to evaluate the existing validation suites. We measured the combinatorial coverage measurement of several existing test suites by inferring the input space from both the test cases and the prepared ACTS configuration file. The coverage metrics seem to look better with the inferred input space due to a smaller number of represented input parameter values. For most of the report objects, the input space inferred from the test cases represents only ~60% of values present in the more complete ACTS configuration file. Despite millions of documents, about 40% of values and equally large percentages of combinations were not included in existing validation. Table III describes the results of the scenarios used to measure the various test suites for the Adobe Analytics reporting engine.

In addition to these analyses, we measured the combinatorial coverage of the 28 million requests used to generate the ACTS config file. The results show an increase in coverage, but the increase is not proportional to the two-order magnitude increase in number of test cases. This provides additional perspective by showing that the current validation would never achieve an acceptable balance of practicality and coverage even if allowed to execute infinitely. However, this analysis does interestingly confirm that the current validation does what it was designed to do quite well. That said, it obviously leaves room for improvement in more ways than one.

TABLE III. COMBINATORIAL COVERAGE MEASUREMENTS

Regression Test Suite (Replay)				
Report Object	***Inferred Input Space***		***ACTS Config***	
	2-Way	***3-Way***	***2-Way***	***3-Way***
Report type: Ranked	25 %	8 %	9 %	1 %
Report type: CA	98 %	96 %	18 %	3 %
Option: Metrics	9 %	1 %	< 1 %	0 %
Option: Filters	2 %	< 1 %	1 %	0 %
Option: Breakdowns	5 %	< 1 %	1 %	0 %
Option: Aggregates	38 %	25 %	1 %	1 %
Integration Test Suite (Quince)				
Report Object	***Inferred Input Space***		***ACTS Config***	
	2-Way	***3-Way***	***2-Way***	***3-Way***
Report type: Ranked	16 %	3 %	8 %	1 %
Report type: CA	65 %	39 %	13 %	3 %
Option: Metrics	12 %	1 %	1 %	0 %
Option: Filters	12 %	1 %	1 %	0 %
Option: Breakdowns	23 %	6 %	4 %	2 %
Option: Aggregates	27 %	9 %	1 %	1 %

V. CONCLUSIONS

These results expose glaring gaps in combinatorial coverage for a validation approach typically viewed as quite comprehensive and certainly sufficient. The combinatorial coverage measurements clearly show that current validation does not provide sufficient coverage. Furthermore, current validation approaches are only capable of marginal increases in combinatorial coverage. Consequently, we reported these results to the engineers responsible for the reporting engine who instantly recognized the deficiency and the potential to drive improvement with these combinatorial coverage measurements. The engineers are considering two applications of the measurements: (1) use CCM to dynamically determine stopping criteria for existing validation or (2) use CCM to create covering arrays to supplement existing validation by generating test cases for missing combinations.

Disclaimer: Products may be identified in this document, but identification does not imply recommendation or endorsement by NIST, nor that the products identified are necessarily the best available for the purpose.

REFERENCES

[1] D.R. Kuhn, R.N. Kacker, and Y. Lei, "Introduction to Combinatorial Testing," Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series, Taylor & Francis, 2013.

[2] L. Ghanderhari, M. Bourazjany, Y. Lei, R. N. Kacker, R. Kuhn, "Applying Combinatorial Testing to the Siemens Suite" 6th IEEE International Conference on Software Testing Proceedings, April 2013

[3] R. Kuhn, I. D. Mendoza, R. N. Kacker, Y. Lei "Combinatorial Coverage Measurement Concepts and Applications" 6th International Conference on Software Testing Proceedings, April 2013

[4] R. Kacker, R. Kuhn, "Automated Combinatorial Testing for Software (ACTS)", National Institute of Standards and Technology, https://csrc.nist.gov/Projects/Automated-Combinatorial-Testing-for-Software

[5] R. Kacker, R. Kuhn, Z. Ratliff, D. Yaga, "CCMCL: Command-line Combinatorial Coverage Measurement tool (CCM) supporting constraints", https://github.com/usnistgov/combinatorial-testing-tools

[6] R. Smith, D. Jarman, R. Kuhn, R. Kacker, D. Simos, L. Kampel, M. Leithner, G. Gosney, "Applying Combinatorial Testing to Large-scale Data Processing at Adobe", 2019, 8th International Workshop on Combinatorial Testing