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

Applying Combinatorial Testing to Large-scale Data Processing at Adobe

Riley Smith
Adobe Inc.
Utah, USA
rilsmith@adobe.com

Darryl Jarman
Adobe Inc.
Utah, USA
djarman@adobe.com

Richard Kuhn
NIST
Maryland, USA
d.kuhn@nist.gov

Raghu Kacker
NIST
Maryland, USA
raghu.kacker@nist.gov

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

Ludwig Kampel
SBA Research
Vienna, Austria
lkampel@sba-research.org

Manuel Leithner
SBA Research
Vienna, Austria
mleithner@sba-research.org

Gabe Gosney
Adobe Inc.
Utah, USA
gosney@adobe.com

***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. This high dimensionality makes validation difficult or intractable. Due to increasing attention from both industry and academia, combinatorial testing was investigated and applied to improve existing validation. In this paper, we report the practical application of combinatorial testing to the data collection, compression and processing components of the Adobe analytics product. Consequently, the effectiveness of combinatorial testing for this application is measured in terms of new defects found rather than detecting known defects from previous versions. The results of the application show that combinatorial testing is an effective way to improve validation for these components of Adobe Analytics. In addition, we report the details of the input parameter modeling process and test value selection to provide more context for the problem and how combinatorial testing provides the structure to improve validation for Adobe Analytics.**

Keywords- Combinatorial Testing, 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. The available configurations result in high dimensionality for any elements of the system that interact with the collected data. For example, the collected data are the main input parameters for the data collection, compression, and processing components. As the product has evolved, the number of configurable elements has increased to at least a few thousand just for these components. Given this domain knowledge, traditional validation of these components relied on randomly generated values for the data input parameters. 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, faults in these components exposed interactions not covered by this traditional approach. These faults revealed the insufficiency of this existing validation method.

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 at Adobe Analytics to apply combinatorial testing to provide better values for the data collection input parameters.

In this paper, we consequently report an industry application of combinatorial testing to the data collection, compression, and processing components of Adobe Analytics. Intending to improve existing validation, the effectiveness of combinatorial testing is measured in terms of new faults found rather than detecting known defects in previous faulty versions. Initial results of this combinatorial testing application found new faults in each of the subject systems with a small set of test cases. For example, a significant fault was detected in the data compression algorithm by a 2-way test set containing only ~150 tests. Furthermore, the results suggest that combinatorial testing may prove more effective than the traditional random approach.

It is important to note that the subject systems vary in size in terms of lines of code, but all have a large number of input parameters with complex constraints and many possible values. This has two main implications that affect implementation of combinatorial testing: (1) no tool existed for creating covering arrays that supported so many input parameters and (2) the values for the input parameters needed to be minimized.

978-1-7281-0888-9/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSTW.2019.00051

Consequently, we also report the details of the input space modeling process. The approach consists of four main steps: (1) define the input space, (2) define values from inferred input space and boundaries, (3) generate covering arrays, and (4) build test cases from covering arrays and input space model. This approach further reduces the validation complexity of these systems.

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

II. Background

A. Data Collection and Compression

For the analytics product, Adobe provides customers with a software development kit (SDK). Customers use the SDK to instrument their own sites and applications. Depending on implementation, the instrumented applications then send data to the Adobe Analytics data collection pipeline. For a large customer, this can exceed 2000 input parameters. Being largely user-defined, the input parameter values approach practical limits. 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.

B. Data Processing

The processing system reads the compressed files and transforms the data for reporting. Being similar to the collection and compression system, this system uses the same covering arrays for over 2000 parameters but has different constraints.

III. Approach and Setup

Applying combinatorial testing to these subject systems heavily depends on the ability to generate covering arrays for the large number of input parameters and input parameter values. However, no existing tool succeeded in generating the desired t-way covering arrays for such a large number of input parameters and input parameter values. As mentioned, the subject systems can exceed 2000 input parameters with over 300 potential unique values. Given that the number of tests for a greedy algorithm generally grows as

$$v^t \log n$$

where

v = number of possible values for each variable

t = interaction strength or t-way interactions

n = number of variables or parameters

The number of rows in the covering arrays quickly approach impracticality (described in [1]). Minimizing the effect of input parameter values proves crucial to making the application practically possible. To continue, Adobe engineers consequently used domain knowledge of the subject systems to model and minimize the parameters and possible values. With this minimized input space, researchers from NIST and SBA Research generated the covering arrays. The remainder of this section gives the details.

A. Input Space Modeling

Due to the large number of parameters and potential values, the input space for these subject systems is largely unknown. Fortunately, the data collection system temporarily stores the data in database tables. Although We used the description of these tables to infer the input space and parameters. Although critical to the modeling efforts, the database table descriptions could not encapsulate all the constraints of the subject systems and the respective input parameters. Consequently, the input space model was refined over several iterations. Furthermore, the table descriptions provided the data types and sizes which proved useful for selecting input parameter values.

B. Value Selection

Being user-defined, the input parameter values can also seem intractable especially when considering how the values could increase the combinatorics. Consequently, we minimized the input space. Using the database table descriptions again, we minimized the input parameter values by limiting the possible values for each input parameter. The data types and sizes from the table descriptions provided an intuitive way to accomplish this: by using the boundary values of the input parameters. When selecting values for each entry in the covering arrays, three values were used.

- Minimum value. The minimum value for number-based database primitives was easy to determine. For character-based primitives, we minimized both the length and byte sizes.
- Maximum value. The maximum for number-based primitives was also easy to determine. For character-based primitives, we inversely maximized the length and byte sizes. This maximization limits this application of combinatorial testing to 3-way interactions. Any t-way interactions above this contained maximum value that exhausted the memory resources of the machine to execute the test cases.
- Unset. In addition to minimum and maximum values, we also used unset values where not prohibited by the database table description.

In addition to these, two test cases were added to the covering arrays: (1) all input parameters set to minimum values and (2) all input parameters set to maximum values. Future applications may incorporate a third entry in which all values use default and unset values.

C. Covering Array Generation

For this application, we aimed to generate 6-way covering arrays because empirical research suggests that most systems do not require interactions beyond this magnitude to detect faults [1]. Given that the subject systems have a large number of input parameters, this seemed like the best target to avoid

poor results. However, no existing combinatorial test generation tool was able to generate the t-way covering arrays desired due to the large number of input parameters. In particular, our application originally required the consideration of t-way interactions of over 2000 input parameters. To be able to construct the required combinatorial test sets, we deployed a combination of a greedy algorithm together with theoretical constructions of covering arrays, which we briefly describe as follows. For the construction of a t-way covering array for more than 2000 parameters, we constructed a set of *seed arrays*, in this case 2-way up to t-way covering arrays for a smaller number of parameters. We generated these arrays relying on the greedy algorithm, called FIPO, described in [4]. Based upon these *seed arrays* we adopted a theoretical construction (described in [5]), which provides means to construct a t-way covering array for doubling the number of parameters. This construction is based on combinatorial methods, such as iterative multiplication and juxtaposition of the seed arrays.

To facilitate automated construction of such covering arrays, we implemented a script that takes as an input the configuration of the target covering array (i.e. its interaction strength, number of input parameters and values), the maximum number of doublings, and parameters limiting the maximum number of input parameters in constructed seed arrays. The program first finds appropriate parameters for seed arrays. It then repeatedly executes the doubling construction described in [3] until the desired number of input parameters is reached. In each step, it either reuses existing covering arrays or generates them on the fly using FIPO, and finally stores the generated target covering array into a CSV file.

Combining the flexibility of the efficient generation of t-way covering arrays, for a comparatively small number of parameters and the capabilities of theoretical constructions to construct covering arrays for a large number of parameters, it was possible to construct covering arrays for the more than 2000 parameters of interaction strength 2 to 5 where the input parameters can take 3, 7 or 10 values. Last, we would like to mention that we also constructed covering arrays for 20 ten-valued parameters, as a direct computation of the FIPO algorithm [4], so as to provide the means for the verification of smaller input models. All generated t-way covering arrays are available publicly online [6]. To the best of our knowledge this is the first time that such large t-way covering arrays have ever been used in terms of an industrial application.

IV. Application

We used the two main parts of the Adobe Analytics data collection pipeline as our subject systems: (1) Data Collection and Compression and (2) Data Processing. After generating the test cases using the modeling and covering array generation approaches described in the previous sections, the tests were executed against the Data Collection and Compression system. After successfully executing these, the resulting compressed files were used to execute the Data Processing system.

We hoped to start with 2-way interactions and then move to 3-way interactions, and so on, until (1) a fatal fault is detected or (2) testing at the current interaction strength does not detect any fault that was not detected in testing with the previous strength. However, the 4-way interactions exceeded the memory resources of the machine used to execute the test cases even after minimizing the input space. While likely due to the 4-way interaction of maximum boundary values, investigations have not yet concluded whether the system should behave in this manner. Regardless, the successful use of 2- and 3-way interactions discovered previously undetected faults. The remainder of this section details these faults.

A. Data Collection and Compression

As stated, we began with 2-way testing. This did not work immediately as we discovered additional, undocumented constraints while attempting to submit the 2-way test cases to the data collection system. Although simple, finding these undocumented constraints marks unplanned success. Adding these missing constraints will prevent future faults caused by incorrect implementations based on bad documentation.

After successfully writing the 2-way test cases into the database tables, the data collection system exported the columnar data to the compression algorithm. However, the compression algorithm core dumped immediately. Originally thought to be of little significance, further investigation revealed otherwise. The compression algorithm declares a pointer to a bit array used to track the occurrence of input parameter values. This buffer initializes to a predetermined size that facilitates over 2000 active input parameters (i.e. a parameter with a non-null value). However, the 2-way test cases exposed interactions where the number of input parameters exceeded the size of the array thereby causing a buffer overflow. Detecting this fault consequently requires the simultaneous use of more than 2000 input parameters. However, the values for the more than 2000 parameters do not matter. So, many combinations of input parameters could have revealed this fault provided the total number of input parameters exceeded the size of the array. Combinatorial testing provided the formal validation approach that systematically detected this fault. As shown, the compression system could not even accommodate the interactions of higher-order input parameters from the 2-way test set. This first application of combinatorial testing detected a significant fault before any users. Consequently, this initial attempt demonstrated enough success to warrant approval for additional applications of combinatorial testing.

B. Data Processing

After successfully compressing the data, we submitted the compressed files to the data processing system. Consequently, this system uses the same covering arrays with over 2000 parameters. This process took more iterations than the previous application of combinatorial testing despite reusing the output of the previous system. Application of combinatorial testing to this system discovered many more, but less significant faults. It is important to note that these faults were not nor ever would be detected using existing validation for this subject system. Consequently, the application of combinatorial testing still proves responsible for the detection of these faults by providing the formal validation approach that allowed detection.

The faults detected in this system largely related to inadequate data input validation. Similar to the initial issue with the previous subject system, there were several undocumented constraints. Violating these constraints caused test cases to prevent successful execution of the program.

Some of the undocumented constraints were valid. These constraints were documented to avoid faults from future implementation errors. However, other interaction failures were due to invalid input parameter handling. Code was changed to appropriately accommodate these valid parameter interactions. For example, a set of columns were defined in the database tables as varchar, but the processing system expects specific values that it interprets as Booleans. Table I summarizes the detected faults within this system. Again, note that this system uses the covering arrays of over 2000 parameters with three values each.

Table I. PROCESS FAULTS DETECTED BY COMBINATORIAL TESTING

	Fault Descriptions, Causes, and Resolutions		
Description	***t-way***	***Cause***	***Resolution***
Flag-type fields throw error	2	Undocumented value constraint	Update input space model
Event-type fields throw error	2	Undocumented format constraint	Update input space model
Parser throws error (CDS)	2	Undocumented value constraint	Update input space model
Parser throws error (JSON)	3	Undocumented format constraint	Add input validation
Invalid date fields interaction	2	Undocumented value constraint	Update input space model

V. CONCLUSIONS

This initial application of combinatorial testing to these subject systems detected seven previously undetected faults. By finding previously undetected faults, this application shows the great potential of combinatorial testing to improve existing validation for the Adobe Analytics product. Furthermore, the application of combinatorial testing to these subject systems provided the engineers of the Adobe Analytics product to learn valuable lessons by challenging ingrained assumptions. Most notably, these results debunk long-held beliefs that these subject systems are too complex to practically achieve systematic coverage.

Overall, the organization considers this application of combinatorial testing successful enough to warrant continued augmentation of existing validation with additional combinatorial testing. The immediate benefactors of future applications include downstream components that consume the outputs of the subject systems mentioned in this paper. In addition, the results of this application lead us to anticipate increasing yields (i.e. detected faults) from existing applications of combinatorial testing as we increase the interaction strength currently limited by available resources.

ACKNOWLEDGMENT

The research presented in this work was partly funded under the Austrian COMET program (FFG).

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] K. Kleine, and D.E. Simos, "An efficient design and implementation of the inparameter-order algorithm," Mathematics in Computer Science 12(1), 51–67, 2018.

[5] S. Martirosyan, and T. van Trung, "On t-covering arrays," Designs, Codes and Cryptography, 32(1):323–339, 2004.

[6] D. Simos, L. Kampel, M. Leithner, Covering Arrays, SBA Research 2018, https://matris.sba-research.org/data/adobe/