2019

Metamorphic relations for detection of performance anomalies

Owen Johnston
Adobe Systems Inc

Darryl C. Jarman
Adobe Systems Inc

Jeffrey Berry
Adobe Systems Inc

Zhi Q. Zhou
University of Wollongong, zhiquan@uow.edu.au

Tsong Yueh Chen
Swinburne University of Technology, tychen@swin.edu.au

Metamorphic relations for detection of performance anomalies

Abstract

Metamorphic relations can be used to improve performance testing by comparing successive runs of the software under test. We examine one such metamorphic relation for page load times, which we used to discover and repair a race condition in the Adobe Experience Platform Launch Tag Manager. Histograms for page load times had different modalities, which alerted us to the presence of the bug. We discuss the need for performance measures in addition to the popular mean and standard deviation. We describe two approaches to automatically determine modality: Gaussian Mixture Models and the Silverman Test for Multimodality. Metamorphic relations which involve these performance measures can be used to alert engineers to the presence of performance anomalies.

Publication Details

Johnston, O., Jarman, D., Berry, J., Zhou, Z. & Chen, T. Yueh. (2019). Metamorphic relations for detection of performance anomalies. Proceedings - 2019 IEEE/ACM 4th International Workshop on Metamorphic Testing, MET 2019 (pp. 63-69). United States: IEEE.

Metamorphic Relations for Detection of Performance Anomalies

Owen Johnston, Darryl Jarman, Jeffrey Berry
Adobe
3900 Adobe Way
Lehi, UT 84043, USA
Email: {ojohnsto, djarman, berry}@adobe.com

Zhi Quan Zhou*
School of Computing
and Information Technology
University of Wollongong
Wollongong, NSW 2522, Australia
Email: zhiquan@uow.edu.au

Tsong Yueh Chen
Department of Computer Science
and Software Engineering
Swinburne University of Technology
Hawthorn, VIC 3122, Australia
Email: tychen@swin.edu.au

***Abstract*—Metamorphic relations can be used to improve performance testing by comparing successive runs of the software under test. We examine one such metamorphic relation for page load times, which we used to discover and repair a race condition in the Adobe Experience Platform Launch Tag Manager. Histograms for page load times had different modalities, which alerted us to the presence of the bug. We discuss the need for performance measures in addition to the popular mean and standard deviation. We describe two approaches to automatically determine modality: Gaussian Mixture Models and the Silverman Test for Multimodality. Metamorphic relations which involve these performance measures can be used to alert engineers to the presence of performance anomalies.**

Keywords: Metamorphic testing, metamorphic relation, performance testing, oracle problem, race condition, verification, validation.

I. Introduction

Performance testing is a common and important task in software development. This paper will address a use of *metamorphic testing* (MT) [1], [2] in the context of performance testing. Specifically, we discuss the need for more sophisticated measures that support retention of information and automated detection of performance anomalies.

One challenge of MT is finding good *metamorphic relations* (MRs) [3], [4], [5]. The task of identifying MRs can be made easier if we have more comprehensive, repeatable and automatable measures for the system under test. For performance testing, timing in terms of mean and standard deviation are often the only measures used. While simple and common, these measures contain partial information and may be easily misinterpreted. First, mean (μ) and standard deviation (SD) are only meaningful for data with normal distributions. But, timing data rarely follows a normal distribution. Secondly, there are some useful information measures other than times.

With the inclusion of measures of frequency clusters and outlier distributions, we can develop additional relations based on the other useful information, such as code paths executed, synchronous vs asynchronous performance and environmental stability to name only a few.

*Corresponding author.

II. Metamorphic testing

In software testing, an *oracle* is a mechanism, or method, through which a tester can decide whether the test case execution results are correct [6]. The *oracle problem* refers to situations where an oracle is unavailable or is too expensive to be applied. For example, big data analytics software is difficult to test because of the lack of an oracle [7]. Similarly, search engines are also difficult to test [8], [9].

Metamorphic testing is a property-based testing method that can effectively alleviate the oracle problem. In MT, instead of focusing on the verification of each individual output of the software under test (SUT), for which an oracle might not be available, testers check the expected relations among *multiple* executions of the SUT (that is, metamorphic relations, or MRs). MRs are necessary properties of the intended functionality of the software. Therefore, if an MR is violated when the SUT is tested on certain inputs, the SUT must be at fault.

MT was originally proposed as a verification technique [1], [2], which could be used by both software development organizations and end-user programmers [10]. It was later extended into a unified framework that covers verification, validation, and other types of software quality assessment [11].

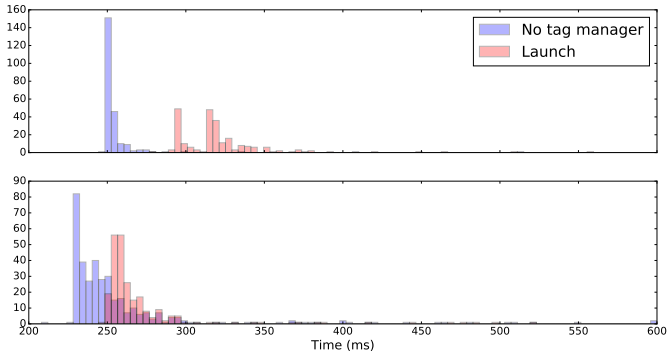

Fig. 1. *Top:* Comparison of page load times for no tag manager vs. Adobe Experience Platform Launch. The Experience Platform Launch histogram (red) shows the presence of a bimodal distribution that was due to a race condition. *Bottom:* The unimodal distribution after the race condition was repaired.

An increasing body of research has investigated the concept of MT for a variety of application domains [5], [12]. MT has been applied not only for conventional software testing but also for other purposes such as cybersecurity enhancement [13], [14] and system understanding [15]. In recent years, there has been a growing trend towards applying MT to test intelligent (especially machine learning) and autonomous systems [16], [17], [18], [19]. In particular, Zhou and Sun combined MT and fuzzing and detected previously unknown fatal software faults in the LiDAR obstacle-perception module of the real-life self-driving car system Baidu Apollo and reported the alarming results eight days before Uber's deadly crash in Tempe, AZ, USA, in March 2018 [20].

Also recently, Segura et al. [21], [22] suggested that MT could be used to reveal *performance* failures. In the present study, we apply MT to the Adobe Experience Platform Launch Tag Manager software in the context of performance testing.

III. Case Study: Adobe Launch

Many webpages use snippets of javascript code from third-parties to identify visitors, collect information about how people use their site, display ads, and set browser cookies. These snippets of javascript are known as tags. Some websites may use dozens of tags on a single page. Products known as Tag Managers are available to simplify the complexity of using all of these tags simultaneously. A tag manager offers an interface to configure all the tags and generates the appropriate javascript to provide all the functionality configured for the site. This generated code is often referred to as a "container." The tag manager's javascript "contains" all the javascript of the tags and orchestrates the order and conditions by which the tags on the website are executed. In this way all the tags on the page can be bundled together into one or more files and can be added to an HTML document in as few as a single script tag..

For example to implement Adobe Analytics (AA), Experience Cloud ID Service (ECID), and Adobe Audience Manager (AAM) on a page without a tag manager, something like this is necessary within the HTML on the page:

```
<script src="VisitorApi.js"></script>
<script src="AppMeasurement.js"></script>
<script src="DIL.js"></script>
<script>
  //AppMeasurement Object Creation
  s_account="ujsltest"
  s=s_gi(s_account)

  //Visitor Object Creation
  s.visitor = Visitor.getInstance(
  "97D1F3F459CE0AD80A495CBE@AdobeOrg")

  /*
  ...
  ...
  Additional AppMeasurement Object
  Configuration
  ...
  ...
  */

  //DIL Object Creation
  var scDil = DIL.create({
    partner: "unifiedjslab",
    visitorService:{namespace:
    "97D1F3F459CE0AD80A495CBE@AdobeOrg"},
    containerNSID:0,
    uuidCookie: {
      name:"aam_uuid",
        days:30
      }
  })

  //DIL+AppMeasurement Integration
  DIL.modules.siteCatalyst.init(s,scDil,
  {names:['pageName','campaign','channel',
   'state','zip','products','server',
   'prop1','eVar1']})
 s.t()

  //Make tracking call
  s.track()
<script>
```

In contrast, when using Adobe Platform Experience Launch, the HTML implementation is greatly simplified, with all the setup and configuration being included within a single library:

```
<script src="//assets.adobedtm.com/staging/
launch-EN1fb320484d8e4419b04396f7de7696d0-
development.min.js"></script>
```

Adobe Experience Platform Launch is the most recent tag manager available as part of the Adobe Experience Platform. Experience Platform Launch enables developers to create "extensions" that customers can install on a web property through an app-store-like experience. These extensions integrate a developer's tag into the Experience Platform Launch container and provide an interface to configure and deploy the tag.

One benefit of using a tag manager is the possibility of improving page load performance by streamlining the process of downloading and executing tags. Some tag mangers, including Experience Platform Launch, can even reduce duplicated code by allowing tags in the container to use shared code that provides common functionality. However there is also the possibility that using a tag manager could increase load times on your site. After all, the tag manager "container" itself includes code that must be downloaded, parsed, and executed.

Bug discovery during quality assurance

As part of the quality assurance process for Experience Platform Launch, we wanted to test whether the overhead of the tag management container would degrade the performance

of page load times. To do this we created test pages that implemented Adobe tags by putting them directly on the page as well as test pages that implemented the tags through Experience Platform Launch. We could then compare the load times of the pages to determine the impact of using Experience Platform Launch. We expected that the two pages would be functionally identical, but their page load performance would probably be different.

Our test environment consisted of web pages hosted on Azure and a desktop machine that accessed these test pages over the internet through Google Chrome. The page load timings were measured using Google Lighthouse.

The same tests were run many times and then averaged to filter the effects of variables that were beyond our control. These uncontrollable variables include server-side performance and network variability. One of the tests compared performance without tag manager, and with tag manager (Experience Platform Launch) with three Adobe products:

- Adobe Analytics (AA) - Adobe Analytics provides reporting, visualizations, and analysis of Customer Data that allows customers to leverage their data in the decision making process.
- Experience Cloud ID Service (ECID) - The ID service provides a universal, persistent ID that identifies your visitors across all the solutions in the Experience Cloud.
- Adobe Audience Manager (AAM) - Audience Manager helps you manage your data pipeline. The service is a catalyst that transforms generic users and raw data signals into actual audience segments used for multi-channel marketing efforts.

One performance metric we tested was "time to first paint," defined as the time between when a user initiates a page load and when the first pixel of the page is painted to the screen. The average time to first paint of these two pages was very similar, with the difference being less than 70ms. However, viewing the histogram of the time to first paint for the two pages revealed something unusual.

While we expected variance between test runs due to the uncontrollable variables mentioned previously, we did not expect differences in the variance between implementations, as both implementations are subject to the same uncontrollable variables. We therefore expected that the histograms should have similar distributions of results, with one histogram being effectivelty translated or shifted horizontally from the other to indicate a difference in average performance. Indeed, this was the case for all the test pages that implemented various combinations of Adobe tags; two unimodal distributions, one a horizontal translation of the other. This can be described as the following metamorphic relationship:

In the domain of Javascript Tags
the following metamorphic relation(s) should hold

- Consistent Modality:
  if a set of 1 or more tags is implemented on a page through the use of a tag manager and on a page without the use of a tag manager
  then the distribution of load times for each page should have the same modality.

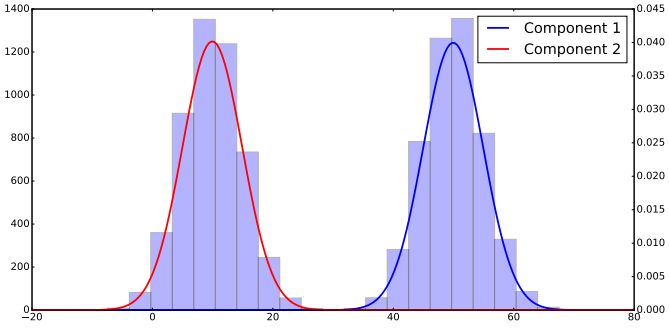

Fig. 2. GMM fits 2 components on a bimodal data set.

For the specific combination of AA+ECID+AAM, the without tag manager scenario had a unimodal distribution, but the Experience Platform Launch scenario had a bimodal distribution (top of Figure 1). This discrepancy or violation of the expected metamorphic relation was indicative of an underlying functional bug.

TABLE I
MEANS AND STANDARD DEVIATIONS OF TIME TO FIRST PAINT BEFORE AND AFTER FIXING THE RACE CONDITION BUG.

	No Tag Manager	Adobe Launch
Before bug fix	μ: 255.3; SD: 21.4	μ: 324.7; SD: 53.2
After bug fix	μ: 266.3; SD: 87.9	μ: 279.9; SD: 89.5

Upon further investigation, it was discovered that a race condition was introduced when implementing this set of tags through the tag manager. This scenario consists of the following sequence of events: (1) the AA and AAM extensions are individually loaded, (2) the AAM extension calls a shared function from the AA extension, then (3) the AA extension sends its network request to track the pageview. However, depending on which library finished loading first, the AAM extension was not always able to call the shared function before AA sent its tracking request. The bimodal distribution was the result of two unimodal distributions being merged together; one for the case when AAM was able to successfully call the shared function before the AA request was sent, and one for the case when function was called after the request was sent. The modality difference of the distributions was therefore an alarm for the performance anomaly. Furthermore, this modality difference provides hints on what kind of possible bugs are being made. In other words, our metamorphic relation not only detects the presence of bugs but also helps to debug them. It is notable that this was a functional bug discovered through analysis of performance testing data.

After fixing this race condition bug, the performance anomaly disappeared, that is, the performance histogram for the tag manager scenario showed a unimodal distribution as

expected, as shown in the bottom of Figure 1. Table I shows the means and standard deviatons of the page load times before and after fixing the race condition.

IV. Measuring modality automatically

One important lesson learned from applying MT to performance testing was the recognition of the need for more performance measures from different perspectives. Mean and standard deviation of timings are typically the only measures used in performance testing. As shown in the above study, if only these measures were used, the race condition problem might not have been detectable.

In order to define good metamorphic relations we need to identify good measures which will both retain valuable information and be computable on the generated data sets within the constraints of testing time and available compute resources. As such Adobe has expanded its performance measures to include two frequently overlooked aspects: (1) modality information and (2) outlier information. By computing and retaining values for these measures, crucial information will be available for future metamorphic testing and the creation of new metamorphic relations.

Modality indicates periodic changes of values. Unless performance testing is carefully controlled many different code paths will be executed during testing. Each path will have a characteristic execution time. Many of these times will overlap and it will be very difficult to determine what path is responsible for a change in performance. It is often not possible to control all execution variables. In this case the use of machine learning techniques can help determine if there is a change in the distribution of performance across many different code paths. In general each mode can be thought of as a group of performance equivalent code paths. If the total number or magnitude of the modes change between performance testing phases this can be a critical alarm of problems. By providing a measure of modality, an alarm (which is technically a violation of the relevant metamorphic relation) can be automatically flagged and presented to the engineer for detailed investigation. Based on our finding we propose a more generalized form of the metamorphic relation we tested that may warrant further research:

In the domain of Web Page Performance
the following metamorphic relation(s) should hold

- Consistent Modality:
  if two pages implement the same functionality
  then the distribution of load times for each page should have the same modality.

In addition to modality, preserving information about outliers is also important. Many problematic issues of performance could be revealed with reference to the information related to outliers. Therefore we further explored the use of various outlier techniques both for the perseveration of data and as a means improving modality prediction.

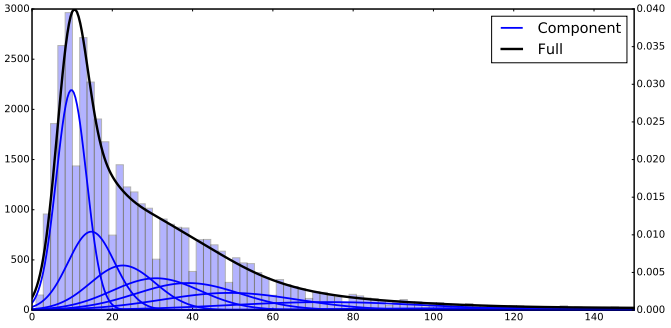

Fig. 3. An example data set where GMM shows pathological behavior by using many components to fit the shape of the skew, although the data is clearly unimodal. The GMM components are plotted in blue, and the full model is plotted in black.

A. Gaussian Mixture Models

The first approach we used for finding multiple modes in performance data was Gaussian Mixture Models (GMM). GMMs are widely used in statistics and machine learning due to their intuitive nature and relative ease of interpretability. As the name suggests, a GMM consists of a set of Gaussian or Normal distributions $\mathcal{N}(\mu, \sigma^2)$, each parameterized by a mean μ and variance σ^2. These individual Gaussian components are then mixed using a Categorical distribution $\phi_{(k=1\ldots K)}$ of size K where K is the number of components. The probability at any x can then be written as

$$p(x) = \sum_{k=1}^{K} \phi_k p(x \mid \mu_k, \sigma_k^2)$$

The Expectation-Maximization algorithm is a common way to estimate parameters for a GMM. We used the implementation in the popular open-source python library Scikit-learn [23]. When estimating the parameters for a GMM, the number of components K must be predetermined. In order to find the optimal K, we fit multiple GMMs with $K = 1 \ldots 20$. We picked the model with the lowest Akaike Information Criterion (AIC) [24], which finds the best fitting model while penalizing those with more parameters, thus preventing overfitting.

After selecting the model with the lowest AIC, we can see how many components best fit our data. For unimodal data, we expect to see a single component in the GMM. Where multiple modes are found, we can examine the parameters of the individual components to determine whether there are unexpected modes, and reason about their causes, as illustrated with an example in Figure 2.

B. Silverman Test for Multimodality

There is a major weakness using GMMs to determine the modality of the data set in an automatic fashion, due to skewed distributions. In many data sets, the underlying assumption that modes will be well-behaved Gaussian distributions is often not true. Timing data is a common example of this, where the data cannot have negative values, which often leads to a skewed unimodal distribution with a long right tail. A GMM may have

a lower AIC by fitting many smaller Gaussian components to account for the shape of the skew, even though the data has a single mode, as shown in Figure 3. For this reason we also implemented the Silverman Test for Unimodality [25].

The Silverman Test works on a notion similar to GMM, called Gaussian Kernel Density Estimation (KDE). In KDE, we replace each observation with a Gaussian kernel function $K(x_i, h)$, i.e. a Gaussian distribution with the mean on the observation x_i, and a pre-determined variance or bandwidth h. We then sum the probability density for all kernels to get an estimate of the probability density function at point y:

$$p_K(y) = \sum_{i=1}^{N} K\left(\frac{y - x_i}{h}\right)$$

The Silverman test uses an automated bootstrapping procedure to find the critical bandwidth h, which is the smallest value that meets a significance threshold as described in Silverman's paper [25]. We then draw samples of the same size as the data set from the KDE model and count the maxima. We repeat this multiple times (100 times in our experiment), and count the percentage of them that have more than one maximum. If the percentage is greater than a significance threshold, we reject the null hypothesis of unimodality. We used an adaptation of Johnsson's python implementation https://github.com/kjohnsson/modality [26].

V. Measuring Outliers in Performance Testing

Calculating measures on performance data can be very sensitive to outliers, which skew simple measures such as mean and variance. Both GMM and the Silverman Test are also sensitive to outliers. For example a GMM will often fit extra components to deal with outliers that can interfere with interpretation of the model. However, in performance testing, the presence of outliers is often of interest as they can signify edge conditions or other unforseen problems. Therefore having robust methods for outlier detection is important for two reasons: first for removing them to get better measures, and second to retain them for subsequent analysis and comparison. We propose using a combination of Windsorization, Interquartile Range, and z-scores for outlier detection.

A. Windsorization

Windsorizing a data set consists of choosing a percentile range and setting data points outside of that range to the values on the edges of the range. For example, choosing the range to be from the 5th percentile to the 95th percentile would mean data points below the 5th percentile are set to the value of the 5th percentile, and likewise data points above the 95th percentile are set to the value of the 95th percentile. This proceedure keeps the number of data points intact, as opposed to trimming or truncation, where data points outside the range are dropped.

TABLE II
Comparison of agreement between GMM and Silverman on 271 data sets.

	GMM 1 comp.	GMM 2+ comp.
Silverman Unimodal	6	169
Silverman Multimodal	13	83

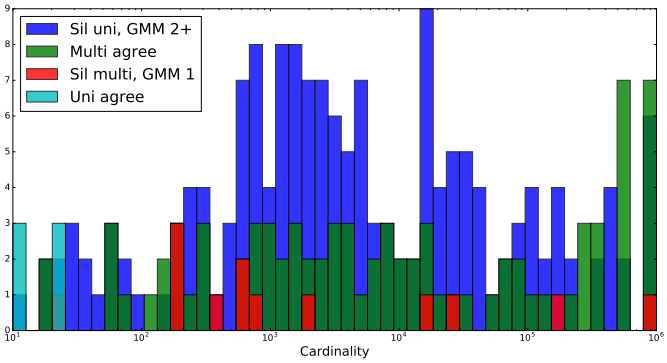

Fig. 4. Histogram of cardinalities of the data sets (log scale), together with results of comparing GMM and Silverman.

B. Interquartile Range

Interquartile Range (IQR) is a trimming method, meaning that data points outside the range are dropped. The IQR is defined as the distance between the 25th percentile to the 75th percentile. Outliers are defined as data points below Q1 - 1.5 IQR or above Q3 + 1.5 IQR, where Q1 is the first quartile or 25th percentile, and Q3 is the 3rd quartile or 75th percentile.

C. z-scores

The z-score centers and standardizes the data set, so that the data set has a mean of 0 and standard deviation of 1. This is calculated for each x using the formula $z = \frac{x-\mu}{\sigma}$, where μ is the mean and σ is the standard deviation of the data set. Once z-scores have been calculated, data points above or below a certain threshold are dropped. For example a threshold of 3 (-3 on the negative side) would mean trimming data points that are more than 3 standard deviations above or below the mean.

We also propose the following metamorphic relationship as a potential topic of further research:

In the domain of Web Page Performance
the following metamorphic relation(s) should hold

- Consistent Outliers:
  if two pages implement the same functionality
  then the properties of the outliers of their respective load time distributions should be similar.

VI. Generalization of Modality Detection Methods

In order to compare the two approaches for estimating modality, we collected 271 performance data sets. The data

sets consisted of timing measurements such as page load time for a diverse set of Adobe Analytics products. These measurements where collected automatically as part of the Quality Assurance test suite. All data sets were univariate, and had a large range of cardinalities, from less than 50 observations to over 1 million.

For each data set we fit the GMM model up to 20 components, choosing the model with the lowest AIC as described in section IV-A. We also ran the Silverman Test for each data set as described in section IV-B. Table II shows a comparison for how often GMM and Silverman agreed on the modality of the data set. In the majority of cases, GMM fitted more than 1 component, while Silverman in contrast predicted unimodality. Figure 4 shows a histogram of cardinalities of the data sets, together with results of comparing GMM and Silverman. It is interesting to note that both Silverman and GMM agree on unimodality only with low cardinality. This suggests that GMM often uses extra components to better fit the shape of the distribution even when the shape is unimodal, as shown in Figure 3.

Although the two approaches often do not agree on the modality of the data set, the metamorphic relation between successive runs can be used to detect perfomance anomalies. For instance, a large change in the number of GMM components between test runs could signal a change in performance that should be further investigated, since under the metamorphic relation, we expected the number of components to remain constant. Similarly, a change in the agreement of the two approaches could be of interest.

VII. Discussion and Conclusion

Our study suggests a promising approach towards automated detection of anomalies in performance data using metamorphic relations. The enriched set of measures in addition to the popular mean and standard deviation discussed here can be compared in successive runs of the software under test. These measures allow performance testing to not only identify changes in performance, but to also signal unexpected or unintentional changes in functionality. Reported changes in modality from the GMM fit and Silverman test can be used to alert engineers to new bugs based on unexpected differences in performance. Large changes in number or range of outliers can also be flagged for follow up investigation.

Acknowledgments

This research was supported in part by a linkage grant of the Australian Research Council (Project ID: LP160101691). Thanks also to the anonymous reviewers for their helpful comments.

References

[1] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: A new approach for generating next test cases," Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, Tech. Rep. HKUST-CS98-01, 1998.

[2] T. Y. Chen, T. H. Tse, and Z. Q. Zhou, "Fault-based testing without the need of oracles," *Information and Software Technology*, vol. 45, no. 1, pp. 1–9, 2003.

[3] T. Y. Chen, D. H. Huang, T. H. Tse, and Z. Q. Zhou, "Case studies on the selection of useful relations in metamorphic testing," in *Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC'04)*. Madrid, Spain: Polytechnic University of Madrid, 2004, pp. 569–583.

[4] Y. Cao, Z. Q. Zhou, and T. Y. Chen, "On the correlation between the effectiveness of metamorphic relations and dissimilarities of test case executions," in *Proceedings of the 13th International Conference on Quality Software (QSIC '13)*. IEEE, 2013, pp. 153–162.

[5] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, and Z. Q. Zhou, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys*, vol. 51, no. 1, pp. 4:1–4:27, 2018.

[6] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015.

[7] C. E. Otero and A. Peter, "Research directions for engineering big data analytics software," *IEEE Intelligent Systems*, vol. 30, no. 1, pp. 13–19, 2015.

[8] Z. Q. Zhou, T. H. Tse, F.-C. Kuo, and T. Y. Chen, "Automated functional testing of Web search engines in the absence of an oracle," Department of Computer Science, The University of Hong Kong, Tech. Rep. TR-2007-06, 2007.

[9] Z. Q. Zhou, S. Zhang, M. Hagenbuchner, T. H. Tse, F.-C. Kuo, and T. Y. Chen, "Automated functional testing of online search services," *Software Testing, Verification and Reliability*, vol. 22, no. 4, pp. 221–243, 2012.

[10] T. Y. Chen, F.-C. Kuo, and Z. Q. Zhou, "An effective testing method for end-user programmers," in *ACM SIGSOFT Software Engineering Notes 30 (4), Proceedings of the 1st Workshop on End-User Software Engineering (WEUSE I)*. ACM Press, 2005, pp. 1–5.

[11] Z. Q. Zhou, S. Xiang, and T. Y. Chen, "Metamorphic testing for software quality assessment: A study of search engines," *IEEE Transactions on Software Engineering*, vol. 42, no. 3, pp. 264–284, 2016.

[12] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, 2016.

[13] T. Y. Chen, F.-C. Kuo, W. Ma, W. Susilo, D. Towey, J. Voas, and Z. Q. Zhou, "Metamorphic testing for cybersecurity," *Computer*, vol. 49, no. 6, pp. 48–55, 2016.

[14] N. Mouha, M. S. Raunak, D. R. Kuhn, and R. Kacker, "Finding bugs in cryptographic hash function implementations," *IEEE Transactions on Reliability*, vol. 67, no. 3, pp. 870–884, 2018.

[15] Z. Q. Zhou, L. Sun, T. Y. Chen, and D. Towey, "Metamorphic relations for enhancing system understanding and use," *IEEE Transactions on Software Engineering*, DOI: 10.1109/TSE.2018.2876433.

[16] M. Lindvall, A. Porter, G. Magnusson, and C. Schulze, "Metamorphic model-based testing of autonomous systems," in *Proceedings of the IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET '17), in conjunction with the 39th International Conference on Software Engineering (ICSE '17)*, 2017, pp. 35–41.

[17] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the IEEE/ACM 40th International Conference on Software Engineering (ICSE '18)*. ACM, 2018, pp. 303–314.

[18] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18)*. ACM, 2018, pp. 132–142.

[19] A. Dwarakanath, M. Ahuja, S. Sikand, R. M. Rao, R. P. J. C. Bose, N. Dubash, and S. Podder, "Identifying implementation bugs in machine learning based image classifiers using metamorphic testing," in *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '18)*. ACM, 2018, pp. 118–128.

[20] Z. Q. Zhou and L. Sun, "Metamorphic testing of driverless cars," *Communications of the ACM*, vol. 62, no. 3, pp. 61–67, March 2019.

[21] S. Segura, J. Troya, A. Durán, and A. Ruiz-Cortés, "Performance metamorphic testing: Motivation and challenges," in *Proceedings of the IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)*. IEEE, 2017, pp. 7–10.

[22] ——, "Performance metamorphic testing: A proof of concept," *Information and Software Technology*, vol. 98, pp. 1–4, Jun 2018.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vander-

plas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[24] H. Akaike, "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716–723, December 1974.

[25] B. W. Silverman, "Using kernel density estimates to investigate multimodality," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 43, no. 1, pp. 97–99, 1981. [Online]. Available: http://www.jstor.org/stable/2985156

[26] K. Johnsson, M. Linderoth, and M. Fontes, "What is a unimodal cell population? using statistical tests as criteria for unimodality in automated gating and quality control," *Cytometry Part A*, vol. 91, no. 9, pp. 908–916, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cyto.a.23173