2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)

Perception and Practices of Differential Testing

Muhammad Ali Gulzar
Department of Computer Science
University of California, Los Angeles
gulzar@cs.ucla.edu

Yongkang Zhu
Google Inc.
ykzhu@google.com

Xiaofeng Han
Google Inc.
xihan@google.com

***Abstract*—Tens of thousands engineers are contributing to Google's codebase that spans billions of lines of code. To ensure high code quality, tremendous amount of effort has been made with new testing techniques and frameworks. However, with increasingly complex data structures and software systems, traditional test case based testing strategies cannot scale well to achieve the desired level of test adequacy. *Differential (Diff) testing* is one of the new testing techniques adapted to fill this gap. It uses the same input to run two versions of a software system, namely *base* and *test*, where *base* is the verified/tested version of the system while *test* is the modified version. The output of two runs are then thoroughly compared to find abnormalities that may lead to possible bugs.**

Over the past few years, differential testing has been quickly adopted by hundreds of teams across all major product areas at Google. Meanwhile, many new differential testing frameworks were developed to simplify the creation, maintenance, and analysis of diff tests. Curious by this emerging popularity, we conducted the first empirical study on differential testing in practice at large scale. In this study, we investigated common practices and usage of diff tests. We further explore the features of diff tests that users value the most and the pain points of using diff tests. Through this user study, we discovered that differential testing does not replace fine-grained testing techniques such as unit tests. Instead it supplements existing testing suites. It helps users verify the impact on unmodified and unfamiliar components in the absence of a test oracle. In terms of limitations, diff tests often take long time to run and appear to generate noisy and flaky outcomes. Finally, we highlight problems (including smart data differencing, sampling, and traceability) to guide future research in differential testing.

***Index Terms*—Testing, empirical study, differential testing**

I. Introduction

High code quality standards demand rigorous testing of large scale codebases which include enormous number of test cases. As codebases evolve, test suites must also evolve, sometimes, requiring hundreds of additional unit and integration tests. Figure 1 illustrates the rapid growth of the total number of tests at Google over one month. Testing at this scale may lead to several critical problems. First, intuitively constructing test cases to cover every corner case is difficult. Second, defining test oracle manually requires the deep understanding of both business and coding logic. Third, running large number of test cases can

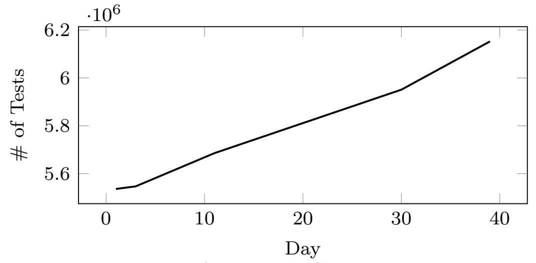


Fig. 1: Number of tests at Google over a month

be time consuming. Last, common testing practices lack the capability to test a software system end-to-end .

Differential testing, also known as back-to-back testing (or diff testing), is a testing technique that attempts to overcome the challenges of generating test oracle and testing real production pipelines [10]. It executes two versions of a software system, *base* and *test*, on the same test input and compares resulting outputs to identify unexpected behaviors. In a diff test, *test* is a modified system that needs to be tested whereas *base* is a verified version of the system that guarantees to produce a correct output. The comparison of intermediate and final outputs allows users to intuitively see changes in system behavior due to codebase changes. These output differences may also reflect on any unwanted behavior due to code bugs in a program. Diff testing often consumes large-scale test data sampled from a production data. Due to large sample size, it significantly increases the chances to exercise all program paths. However, it is still challenging to perform data comparison in order to isolate change impact and to sample the data while achieving full code coverage.

Given these known trade-offs (also described elsewhere [10]), diff testing is widely used in practice where it supplements traditional testing methods (*e.g.,* unit tests and integration tests) with the primary goal of verifying preserved behavior of a modified system. To our best knowledge, differential testing has never been studied as a pure testing technique and how it is leveraged in practice. At Google, diff testing has already been adopted by hundreds of teams with over 600 active diff testing framework users.

In this paper, we present the findings of the first user study on diff testing in practice (*i.e.,* at Google). We conducted this user study in three parts: (1) an online survey completed by 117 software engineers spanning across more than 100 teams, (2) additional one-on-one

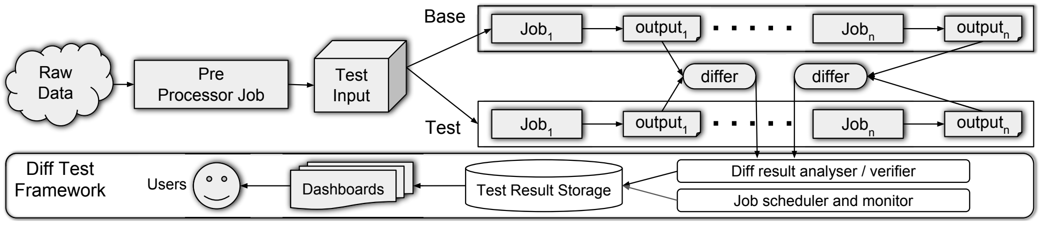


Fig. 2: Diff test workflow and framework

interviews with 5 experienced diff test users, and (3) log analysis of 2104 diff tests from an organization-wide diff testing framework. Through this investigation, we aim to understand the common practices and configurations of diff test and the benefits users acquire by using it. We also seek to find circumstances where conventional testing practices are ineffective and the features that make diff tests useful. Most importantly, we plan to discover the challenges and limitations of using diff tests. The survey is further extended with few open-ended questions to collect general perception, experience, and future needs and improvements desired in diff testing. The research study questions are summarized below:

- **RQ1 Diff Test Practices**. How do software developers practice diff testing? *i.e.,* the input and output data, structure and configurations of diff tests.
- **RQ2 Usage and Popularity**. What are the usage patterns of diff tests? *i.e.,* the frequency and the breakdown of running time of diff tests?
- **RQ3 Diff Test Incentive**. What are the key benefits and features of diff testing?
- **RQ4 Desired Improvements**. What are the major challenges and limitations in using diff tests?

Based on our investigation we show that differential testing is not perceived as an ultimate testing solution to solve all testing needs. Instead, it supplements fine grained tests (such as unit test) with an additional coarse-grained verification. The responses obtained from interviews reflect that diff tests are highly preferred for end-to-end testing due to its low maintenance cost. It often consumes large input dataset aiming for higher test coverage, simulates a real production system, and does not require a test oracle upfront. In some cases (such as web UI testing or code migration testing), diff test is easy to construct and it produces actionable outcome. Similarly, it has been commonly used to detect the impact of a change on other unmodified modules within the same system. Therefore, it has been repeatedly referred as a sanity check.

In terms of limitation, many factors can introduce flakiness in diff test outcome. We observe that diffs heavily rely on a smart differencer which is capable of isolating a problem and filter noise. Test data sampling is challenging and smart sampling techniques can benefit diff testing. Our analysis of 2104 diff tests reveals that majority of diff tests take long time to finish in which a large portion of the time is spent on differencing the output of jobs. Finally, our findings put emphasis on future research towards developing better data differencing and sampling techniques.

This paper presents the first study to the best of our knowledge that empirically explores differential testing in practice. The rest of this paper is organized as follows. The following section provides some background knowledge of the testing practices. Section III discusses the methodology of this user study. Section IV presents the results of study followed by discussion in Section V. Section VI presents related work and Section VII concludes the paper.

II. Background

A. *Differential Testing in Practice*

Differential testing is a testing paradigm which also inspired other testing techniques e.g., regression testing, mutation testing and N-version testing [19]. It targets complex software systems that are rapidly changing. The fundamental goal of differential testing is similar to that of regression testing, i.e., detecting the bugs introduced by any of the recent modifications. While regression testing requires modifications to test suites to accommodate any new change in the codebase, diff testing verifies (1) the behaviors that are intended to be unchanged remain unchanged, and (2) all the differences in the output are either expected or explainable. To verify this, diff test requires comprehensive data comparison to find all differences in the outputs coming from updated and original version of a program. This is in contrast to updating the current test case suites in unit testing or integration testing.

As shown in Figure 2, a typical diff test consists of three steps. Firstly, a test input is either sampled from data in production environment or generated synthetically with the goal to achieve comprehensive test coverage. Most diff tests create a temporary storage for the test data, and perform some data pre-processing (e.g., transforming the raw data into correct format, cleaning up unused and noisy fields). Second, two versions of a software system are selected. A *base* version is chosen with the assumption that it is bug free, while the other is a *test* version which is modified and, thus, needs testing. Both versions of the system are executed with the same input data prepared in the first step. This step is called testing phase. In the last step, i.e., output diffing, intermediate and final output from both *test* and *base* runs are collected and compared. If the output is expected to be preserved, a

Section	Q #	Survey Question	Response Format	Free-Text Option
1 (Diff Test in General)	1	How many Diff tests does your team have?	MCQs	✗
	2	How does your team use Diff test?	MCQs	✓
	3	Do you find Diff tests useful?	1 to 10 Scale	✗
	4	What are the interesting things you like about your Diff tests?	MCQs	✓
	5	What are the pain points of Diff tests?	MCQs	✓
	6	What are the useful features of end-to-end Diff testing framework of your choice?	MCQs	✓
	7	What are the new features you think would userful?	MCQs	✓
	8	Suggestions, recommendations or comment for Diff testing framework designer	Free Text	✓
2 (Questions about a selected Diff Test)	9	How is a particular Diff test structured?	MCQs	✗
	10	What is the base of this Diff test?	MCQs	✓
	11	What is the test of this Diff test?	MCQs	✓
	12	What is the running time of this Diff test?	MCQs	✗
	13	How large is the test input data?	MCQs	✗
	14	What is the sampling strategy (if any) used for Diff testing and where is the data stored?	MCQs	✓
	15	What is the output of the tests that are diff-ed?	MCQs	✓

TABLE I: Online survey questions with their response format

passing test should not produce any difference between the two outputs. In case when changes in behavior are expected, the diff can be analyzed to verify if the changed behavior is correctly reflected in the output difference.

B. Differential Testing Automation

Over the past few years, multiple diff testing frameworks have been developed internally at Google. With the goal of simplifying diff test creation and maintenance, these frameworks require none or very limited amount of programming. Instead users can automate the orchestration of diff tests using a configuration file, which should describe (1) a test data or a data generation program, (2) how to run the *test* and *base* version of a program, and (3) customized differencing rules (if needed). Diff test frameworks will perform these user specified diff tests automatically and output the difference (or delta). Figure 2 also demonstrates a diff test framework in high level, which contains the following components.

- **Job scheduler and monitor**. Job scheduler and monitor are responsible for running the input generation jobs, *base* and *test* systems, and collecting the output of each job pairs for later comparison.
- **Differ**. Differ is critical for any end to end diff testing framework since the output of the two executions can be large, and comparing them manually is error prone. Many commercial or research [15] differencing tools can be integrated within such testing frameworks to highlight the differences.
- **Storage**. All test results, including the status and output of all jobs in *base* and *test* systems, diffing results such as counters and diff samples should be stored in a persistent storage.
- **User Interface** Most diff test frameworks provide a web-based user interface for users to monitor diff test status and analyze diff test result. Aiming to provide one-stop UI for user, some frameworks also integrate their UI with other systems so users can report bugs or trigger the following releases after examining the diff test results.

III. Methodology

We structured our user study in three categories. An online survey distributed among software engineers at Google followed by one-on-one interviews with diff test users. Lastly, we analyzed test logs collected through one of the most popular end-to-end diff test frameworks at Google.

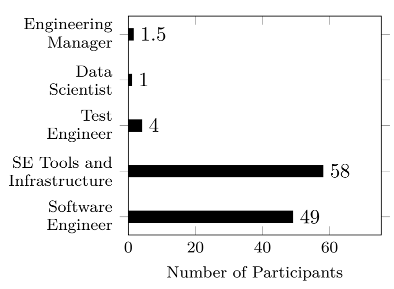
A. Survey questions

We designed survey questions to investigate the weaknesses and strength of diff testing and discover new insights about the perception of diff testing (and testing in general) that can fuel future research on testing. The online survey form comprised of 15 multiple choice questions divided into two sections.

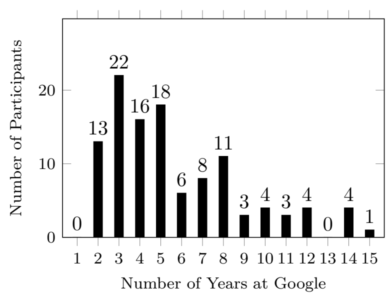
The first section of survey contained 8 questions to understand the overall perception of diff testing. In the second section, we asked participants to select the most important diff test of their team and answer 7 additional questions on the structure, final outcome, running time, input data size, etc., of the selected diff test. A participant could select multiple answers simultaneously and could also enter additional response in a free-text format. Two questions in the survey were free-text format only. We limited the number of free-text only questions to two to maintain higher response rate by avoiding time consuming questions. One question asked the survey participants to answer on a Likert scale. Table 1 presents the most important survey questions. We categorize survey question responses by manually looking at open text format answers and visualize aggregated responses on different chart graph to observe the trends.

B. Subject Interviews

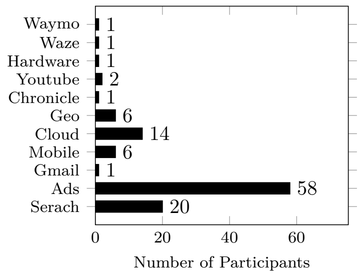
Five interview were conducted with diff test users. Among the subjects volunteered to take part in the interviews, we randomly selected five. The interviews were performed one on one and lasted for 20 to 25 minutes. For the first five minutes we ask warm up questions regarding a subject's job title, team, tenure at Google, and years of experience with diff tests. In the next 15 minutes, we inquired the reasons behind using diff tests and the testing method it replaced. We also asked about effectiveness of diff tests and if it is able to find bugs that other techniques overlooked. In the last 5 minutes, we encouraged participants to share their general thoughts on



(a) Job Titles



(b) Tenure at Google



(c) Teams/Organization

Fig. 3: Demographics of survey participants

diff test and the future improvements they would like to see in diff tests.

C. *Test Log Analysis*

We performed statistical analysis on diff test logs that contained telemetry data from an end-to-end diff testing framework at Google. A log entry in this dataset comprised of running configurations as well as runtime metrics (such as runtime, execution order, job status etc.) of the executed diff test. In total, we analyzed 282423 runs of 2104 diff test projects reaching millions of job runs. Each of such job contains an execution order number and the type of job (testing, diff or counter diff). Using this information, the timeline of each job is re-constructed to ultimately find statistics related to time spent on each type of job as well as other common statistics.

D. *Threats to Validity*

Due to limited number of survey responses and one-on-one interviews, the conclusions derived in this study may not generalize to other settings. As stated before, all participants of this study belong to one organization where a specific culture and bias may hold regarding a specific development style. Google hosts its codebase in a huge monolithic repository [6] that requires extensive pre-commit testing with strict resource and running time requirements as well as data privacy control. Therefore, a more defensive attitude towards testing is preferred to eliminate the possibility of releasing buggy production code. Hence, the results of this study may not generalize to other organizations emphasizing less on testing.

Due to automatic orchestration of diff tests using frameworks, the ease in diff test deployment can impact the user's perception of diff test. We tried to mitigate the effects of such confounding variables by explicitly distinguishing the diff test from framework. However, such biases may still pose threats to internal validity.

The online survey comprised of a few multiple choice questions with predetermined answers based on informal feedback from diff test users. While submitting their responses, study participants may be tempted to select one of the available responses rather than writing their own which may contaminate the study results. We attempted to encourage participants to provide open-ended answers in text.

IV. Survey and Analysis Results

In this section we present survey and analysis results, which are further categorized to answer the four research questions proposed in Section 1.

A. *Demographics*

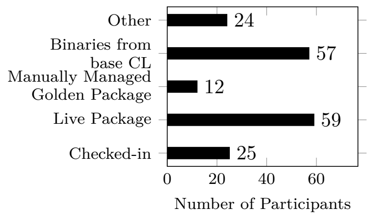
Over a period of 30 days, we had received a total of 117 responses with approximately 19% response rate. The survey responses were confidential but not anonymized. In Q1 of the survey (Table I), users provided their teams and usernames. Surprisingly, the 117 participants of the survey represented 110 different engineer teams. We believe this is due to the fact that most teams rely on a couple of engineers to develop and maintain diff tests. Figure 3a illustrates different roles/job titles of survey participants. Software Engineers and Software Engineer, Tools and Infrastructures are the top two job titles.

On average, the participants have been working at Google for approximately 4.5 years. Figure 3b shows the distribution of participants' tenure at Google. Based on the team names of participants, we derived the nature of the teams' product and their sub-organizations within Google, which is shown in Figure 3c. The highest participation in the survey was seen by Google Ads organization followed by Google Search Platform. The teams involved in the survey develop a diverse range of products including web services, mobile services, data analysis applications, self-driving cars, and cloud services. We believe the survey subject pool to be reasonably diverse with ample representation from each type of software system.

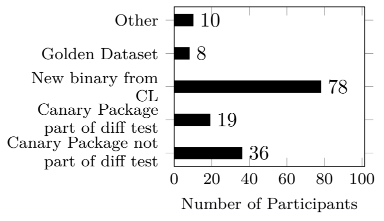
B. *Practices of Diff Tests*

As illustrated in Figure 2, a diff test can be performed in three stages. To understand the common practices of deploying two pipelines for a diff test, we asked in the survey about the structure of diff tests (Q9 - Q15, as listed in Table I).

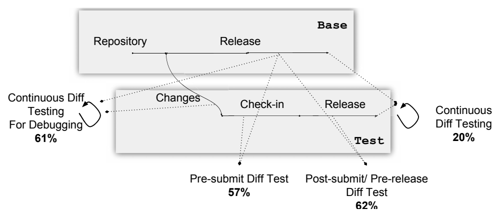
Step 1: Test Data Generation. The primary criteria of test data generation is to achieve high test coverage. However this criteria is hard to measure, so most teams



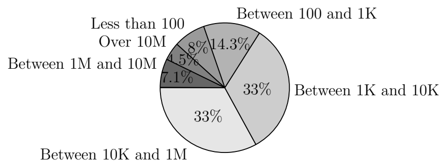
(a) *Base* Side of a diff Test



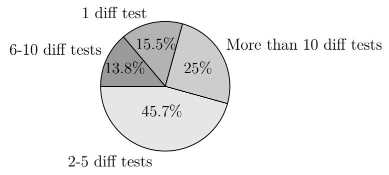
(b) *Test* Side of a diff Test



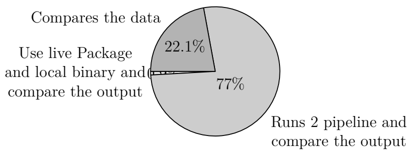
(c) Choice of base and test version of diff tests



(d) Size of the input data



(e) Number of diff tests per team



(f) The goal of a diff test

Fig. 4: Practices and Configurations of diff tests

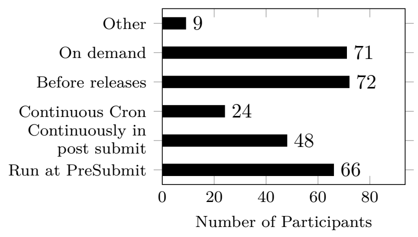
instead use a large amount of input records to achieve the desired test coverage. Responses to Q13 reveal the size of input dataset in terms of number of data records, which ranges from 100 to more than 10 million. Figure 4d shows the response distribution for Q13. Over two-thirds of the participants mentioned that their test input contained from 1000 data records to 1 million records. A small fraction of teams (5 teams) use test data containing more than 10 million data records, further interviews with the these participants showed that their diff tests are used for large scale data processing pipelines and even with low sampling rate below 1%, the totally number of data records can easily go over 10 million.

In Questions Q14 and Q15, we would like to find out how input data was selected and where it was stored. 49 participants (43.4%) mentioned that they did not perform any sampling and simply used the entire dataset. For the participants that did sample the input data, majority of them (49, 72%) used simple sampling strategies like random sampling, range-based sampling, or first 1% data, etc., as the selection strategy. The remaining 19 participants (28%) used various sampling tools developed internally to minimize the test input size. Interviews with the participants showed that these sampling tools use value bucket based sampling algorithms to select input records covering all possible values in each field. The most widely used input data format for diff tests is Protocol Buffers[4], which is *lingua franca* for data at Google between different modules and software systems. The generated test data were mostly stored in shared network storage i.e., SSTable [1], or in distributed data store *i.e.,* BigTable [1], or distributed database *i.e.,* Spanner database [2].

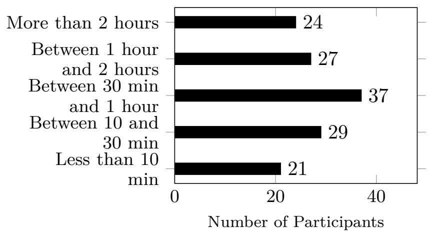
Step 2: Test and Base System Execution. Once the test input data is ready, the next step is to select and run the *base* and *test* versions of a software system. *Base* acts as a ground truth version whereas *test* is the version under test. Survey questions Q10 and Q11 ask how the selected diff test is staged, aiming to gather insight about a standard way (if there is any) of selecting *base* and *test* systems. Responses for these questions are visualized in Figure 4. Starting with the *base*, the majority (70%) of teams use versions that are known to be good; indicated either by being used in production or marked as golden (Figure 4a). Interestingly, around 21% of the participants mentioned that they cache the output of previously known golden version as *base* (similar to test oracle) and directly use that as the output for diffing in next step. Using cached data as a base version is practiced to avoid repetitive runs of the same binary on the same data, which may cause waste of resource and time.

For the *test* version (Q11 and Figure 4b), 60% of the participants selected the binaries built with new code changes (usually called Change List at Google, *i.e.,* CL) as *test*, these code changes can be either under review (pre-submit diff test) or already submitted (post-submit diff test). 36% of them pick "Canary" builds *i.e.,* the builds that passed all other tests and are ready to be released to production environment. In some cases (7%), rather than selecting a program binary, users selected a dataset generated from external pipeline and compared it against the output of *base*. Further interview with participants showed that these outputs are usually selected directly from another staging or testing environment, this is also an optimization strategy to avoid redundant execution of the same *test* binary. Both *base* and *test* systems may comprise of multiple jobs, and each job produces its output that should be collected and compared in the following steps.

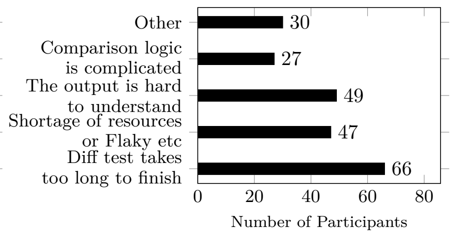
Step 3: Output Differencing. The last step of a diff test is to difference the outputs from *base* and *test* versions of the software system. In Question 15, we inquire about the format of outputs which will be compared later. 80% of the participants mentioned Protocol Buffer format whereas rest of the participants mentioned output dataset in the format of csv, html, image, video, or plain text files.



(a) diff test usage in development



(b) Running time of diff test



(c) Painpoints of using diff test

Fig. 5: Characteristics of diff tests in general.

6% of the participants mentioned that they also perform differencing on Job Counters, which are basic statistics of a certain monitoring variable, such as number of records processed in each category, or intermediate performance metrics, like cpu, ram usage, http response latency, etc., which are captured at runtime.

C. Usage and Popularity

The survey also included questions regarding the usage of diff tests. Q1 of online survey inquires about the number of diff tests each participant's team has (as seen in Table I). 29 responses mentioned that their teams use more than 10 diff tests whereas 18 and 16 responses state the use of just one and 6-10 diff tests respectively. Figure 4e illustrates this result in a pie chart. Question Q2 investigates the point in the development cycle when a diff test is invoked. We provide 4 pre-defined answers based on the development testing procedure at Google. Figure 5a presents the distribution of different diff tests usage. Majority of the participants (60%) responded that they either run diff tests before the release, on-demand for debugging, or before checking in new code. Around 20% responded that they use diff tests in a Cron job fashion to perform regular sanity checks at fixed intervals.

Question Q9 asks about the use case of diff tests to investigate the most popular usage scenario of diff tests. 77% of the participants compare the output of two versions of a program whereas 22% only compare the two datasets in their diff test. Figure 4f shows the distribution of this result. In question Q12 and Q13, we ask participants to provide the running time of the chosen diff test to understand the scale of the test. We observed an approximately normal distribution of total running time of a diff test with the mean time of 30 minutes to 1 hour as seen in Figure 5b. Only 24 participants mentioned their diff tests take more than 2 hours to finish.

We further analyzed the logs of 2104 diff tests running through a company-wide diff testing framework to find the quantitative evidence of usage patterns. Table 6 shows the overall statistics of this analysis. Among 282,423 runs of all diff tests 19601 (6.9%) runs failed to finish, 168901 (59.8%) runs finished with diffs in the output, and the remaining 93921 (33.3%) runs finished without finding any diffs. On average, each diff test contains approximately 9 jobs where 5 are diff jobs and 4 are non-diff jobs, including

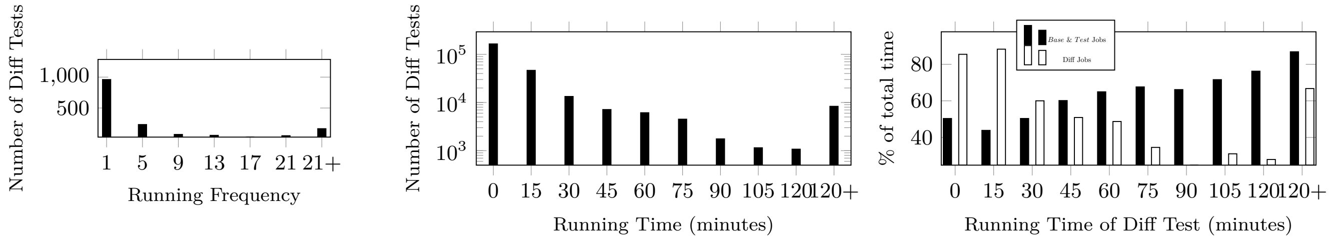
Total diff test projects	2104
Total Diff test runs	282423
Failed test runs	19601
Finished test runs with diffs	93921
Finished test runs without diffs	168901
Average jobs in Diff test	9.85
Average non-diff jobs in Diff test	4.68
Average diff jobs in Diff test	5.17

Fig. 6: *Diff* test log statistics

data preparation jobs, jobs in *base* and *test* systems, etc. In terms of running frequency, as shown in Figure 7a, excluding 538 on-demand diff tests, majority (61%) diff tests run just once or less in a day. Only 12% diff tests run over 20 times a day. We also reconstruct the timelines of these jobs and measured the breakdown of running time of each diff test. Figure 7b presents the number of diff tests (y-axis in log scale) with the corresponding running time (x-axis). Majority of the output differencing jobs take under 15 minutes, however, 4% of the diff tests take between 2 hours and up to 146 hours. We observe an exponential decrease in the number of diff tests as the running time increases. Figure 7c shows the breakdown of running times of diff test instances, note that since many jobs are running in parallel, the combined running time of all jobs always exceeds the actual diff test running time. Short running diff tests that finish within 30 minutes spent most time (63%) in differencing the outputs whereas, for longest running diff tests, a large portion of time is consumed by the execution of *base* and *test* systems.

D. Incentives to Adopt Diff Test

In addition to what traditional testing techniques has to offer, we inquired the benefits attained by using diff test. We also explored specific cases where diff tests perform better than expected. For this purpose, we asked the participants Q4 and Q3. In response to Q4, we received several insightful responses through free-text format which were categorized manually into different groups. 60% of the participants responded that due to large test data diff tests provide high code coverage. Around 53% of the participants mentioned that real production pipeline in diff testing appeals to them the most. 47% mentioned that test output diffs are helpful in debugging a test failure. In



(a) Test frequency in 24 hours (b) Number of diff tests vs running times (c) Running time breakdown of diff tests

Fig. 7: Analysis results of a diff test framework log

Q3 (see Figure 8), 58% of the participants rated diff tests 8 and more in terms of usefulness where as 4 participants rated diff tests 4 or less which provided us valuable insights of when diff test fails to perform (see Section V).

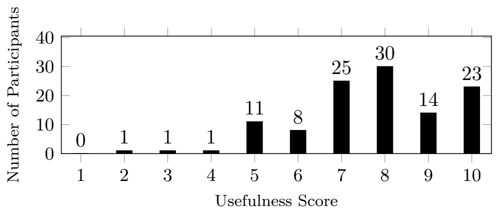


Fig. 8: Usefulness of Diff test on Likert scale where 10 means most useful.

E. Diff Tests Limitation and Future Improvements

Despite diff test's popularity, we also aimed at finding the limitations and challenges of using diff test and the follow-up improvements to guide future research in testing. For this purpose, we explicitly asked online survey participants to list the pain points of using diff tests in Q5. Figure 5c shows the results. 66 participants mentioned that diff tests have slow performance (see Section IV-C) and around 50 participants reported the flaky nature of diff tests mainly due to external dependencies. A few responses reported that due to indecisive nature, the output of the test is hard to understand. The online survey also asked for any desirable features or improvements in diff tests with the aim to highlight software testing challenges, which will be discussed in details in next section.

V. Discussion

In this section, we compile insights by analyzing user responses. Similar to section 4, we categorize the findings under 4 research questions.

A. RQ1 - Diff Test Practices.

We analyze the structure of diff tests from the responses of Q9, Q10, and Q11. Majority of diff tests (78%) are constructed with running *base* and *test* systems and then comparing the output. Unlike unit tests or integration tests which often isolate the system under test (SUT) by removing external dependencies using mocking services, these diff tests try to be as identical to a real production systems as possible. The *base* and *test* systems either directly reuse the configuration of the production system or just apply limited modifications. The remaining 22% diff tests directly compare two sets of data. Interviews reveal that these datasets are some kind of meta-data used in production. The purpose of these diff tests is to verify the consistency in data quality among changing data sets. For example, some participants mentioned that their teams generate a new set of aggregated machine learning training data every week using multiple raw data sources from past 60 days. Before pushing the new dataset out for model training, they use diff tests to thoroughly verify that the changes between new and old datasets are within predefined thresholds. Again, the systems generating these *base* and *test* datasets for diffing are still the real production systems.

Moreover, we did not observe any cases where current unit tests and integration tests are being replaced by diff tests, instead diff tests supplement the existing unit or functional tests. We believe the major reasons are: (1) diff tests take more resources and longer time to run and cannot be executed frequent enough for every code change, (2) privacy control policies at Google forbid using production data on uncommitted code changes, and (3) existing unit and functional tests provide deterministic results that can help engineers to quickly debug and iterate.

B. RQ2 - Usage and Popularity

Among common usage of diff test, we observe that most teams at Google use diff tests as the last line of defense against bugs by testing a complete working pipeline. Diff tests often match the behavior of a large scale integration test to verify if final product is stable. We find that diff tests users do not expect a diff test to provide decisive and informative outcome. Instead, the test is aimed to perform a sanity check before the modifications gets checked in or released. Therefore, pre-built or released binaries are often used as *test* system. Diff test also checks if new changes or modifications have broken hundreds of other features that should not be impacted. As one participant stated in response to Q2:

❞***Response 1.*** *"Diff test is the only sane way to make sure I am not affecting any of the other 100+ features in Search when I change my own."*

Among the responses of Q4 we find that many small teams build interconnected components towards a larger system. Since the developer of a component might not be well versed in external team's component, it can be challenging to develop unit test cases. A diff test can be designed to verify the preserved behavior of a system by comparing the output of *base* and *test*.

❞***Response 2.*** *"We have 6+ internal customer teams that work independently, and they need to know that no one has broken their features when changing a shared platform."*

Half of the responses in Q2 mentioned the use of diff test in "pre-check in" phase to verify that the recent modifications do not adversely affect the features of other modules. Similarly, a response to Q4 mentioned the use of diff tests as a *smoke test.* In *smoke test*, only a few most significant features of a program are tested.

❞***Response 3.*** *"I use them as a very expensive smoke test to verify a CDD [configuration] change."*

In Q1 and Q8, we found that three quarters of the participants' teams have under 6 diff tests and about two-thirds of the teams have diff tests that ingest under 1M data records. We derive two insights from these statistics: (1) diff tests are designed to test the most valuable features instead of every feature of a system and (2) diff tests performs a large scale last minute sanity check before putting the system in production. From interviews, we discovered that diff test input is usually carefully sampled from production data by each team so that it covers the important aspect of each feature and therefore, it evolved over several years as the product grew.

C. RQ3 - Diff Test Incentive.

One of our aims from this study is to investigate the uniqueness and incentives in using diff tests. From responses to Q3 and Q4, we believe that *comprehensive test coverage with low maintenance overhead* and *simulating real production systems* are the main reasons behind the popularity of diff tests at Google. Around half of the participants mentioned that the diff of output is helpful in debugging. We believe that showing a delta in the outputs helps users in bug traceability. One of the participants mentioned that diff tests are really good at highlighting the actual impact of a code change.

❞***Response 4:*** *"It always captures completely unexpected things that you cannot think of. It had helped us find hundred of bugs that other tests missed"*

The unexpected outcomes of a test points to the fact that it is challenging for users to imagine all the possible outcomes of a program. Therefore, users find the changes (*diffs*) extremely useful that are, otherwise, difficult to intuitively construct. Participants mentioned that when dealing with a large codebase or large scale software development, generating assertions for unit or integration testing is extremely challenging. Since diff tests do not presume any knowledge of test oracle of the component under test therefore, writing such tests becomes easy.

❞***Response 5.*** *"We can't write actual assertions because the data is too broad and the "correct" solution is almost impossible to find. diff tests are about our only option for true coverage."*

We believe that such cases are frequent. For instance, one participant used diff test to compare the performance of a modified program with that of a base version. Since performance figures are nearly impossible to find statically, assertions about the performance are difficult to construct.

❞***Response 6.*** *"We like to use diff tests for accurate performance testing."*

We also discover that diff tests have been used in a variety of settings where they are more effective than conventional test technique.

❞***Response 7.*** *"Screenshot diff tests are a much more reliable way of identifying the results of your front end changes than looking at various css diffs."*

In this case, diff test appears to be a suitable technique to test web UI. Such testing is hard to automate because of dynamic content and, real time interactions and environment producing different outcomes. The outputs are web pages in HTML and CSS with boilerplate code which adds noise. Diff tests filters all the styling code and presents the actual comparison of the content.

D. RQ4 - Desired Improvements

Slow performance with long running time to get test results is the most voted pain point for diff tests (according to the responses of Q5). There are three major reasons for this: (1) diff test simulates real production systems and exercises end-to-end pipelines, (2) large scale datasets are used to achieve high test coverage, and (3) diffing jobs usually take long time to finish.

The responses to Q13, presented in Figure 4d, show that most input datasets for diff tests contain between 1000 to 1M data records. While this range does not appear to be large, our interviews clarify that data records are often encompassed in data structures with thousands of fields including nested data structure and collections. Therefore, even a few thousands data records can easily reach a few GBs in size. On the other hand, all diffing jobs at Google are implemented using batch-based Mapreduce technology, and cannot emit partial diffing results while running, therefore users have to wait for the diffing jobs to finish to see the results.

The result of diff tests can be inconclusive and hard to understand. Since there is no definitive fail and pass, users may not receive any actionable insight from test outcome. Manual comparison is a time consuming and error-prone task while automatic comparison is hard to develop but desirable.

❞***Response 8.*** *"Diff tests assume that there is no change and always require a human to evaluate results."*

❞***Response 9.*** *"It can be cumbersome to flag good vs. bad diffs."*

❞***Response 10.*** *"In general, it is difficult to achieve clear pass/fail signal (but it largely depends on the project, not the diffing solution)."*

❞***Response 11.*** *"Very hard to tune to get a clear signal. We've devoted a lot of effort at getting a clear, actionable signal and it requires a lot of domain knowledge."*

Diff tests are often flaky in nature. Since the scope of diff tests span the entire system rather a single component, multiple factors can affect the test outcome and introduce flakiness. According to an empirical study of the root cause of flakiness, factors affecting the flakiness include network, scheduling time, IO, randomness, unordered collections [9]. In our understanding such dependencies are common in many products across the organization.

❞***Response 12.*** *"Flaky diffs caused by non-determinism in the product itself are hard to identify and fix."*

❞***Response 13.*** *"External data/backend dependencies make it hard to be deterministic."*

When the test version of a program introduces a modification that impacts every output, the entire output will be part of the diff. If one of such output is generated by a bug, the overwhelming size of diff may overshadow the diff pointing to the actual bug. This is referred to as noise. Large size diffs lack precision making it easier for the user to overlook the faults in the output diff. This demands a high quality differ that can provide meaningful and actionable diffs.

❞***Response 14.*** *"False positives when new features are added is among the major pain points."*

❞***Response 15.*** *"Non-determinism in a search stack causes "noisy" diffs which are non-essential for the end-users."*

❞***Response 16.*** *"There are no managed technology to control expiration of diff ignores - everybody has to implement their own."*

❞***Response 17.*** *"The results are too noisy and we cannot detect a slow crawl or even a small step regression. we have tried running with A(base) and B(test) both being the same binary and our test results indicated a significant regression..."*

Last but not the least, like all other testing techniques, diff tests also require the availability of input test data and therefore, they inherently suffer from the challenge of input data sampling and selection. Existing sampling algorithms used within Google are either random sampling or value bucket based sampling. They do not consider the correlation between the different data fields or input sources, thus cannot provide guaranteed test coverage. The desired smart sampling algorithm should be able to select or generate minimized input dataset to trigger the changed behavior of a program and produce meaningful different outputs. This is a challenging optimization problem, but can significantly improve the determinism and efficiency of the diff tests.

VI. Related Work

Differential testing was formally introduced by McKeeman [10] as regression testing geared towards large software systems. The work presents important aspects of diff testing in the context of C compiler testing. McKeeman also highlights the issue of oracle identification and testing at scale and shows that diff testing has proven unpopular among the developers working on already tested software. Such findings are contrary to our observations in a large organization setting and, therefore, motivated this user study. Evans et. al. reinvented differential testing by automatically repairing test cases for a modified system and then contrasting the output of two systems [3]. Their tool requires an automated characterization test generator (ACTG) to generate two test cases for each version and then compare the output. ACTG is a non-trivial problem to automate and requires a deep understanding of program semantics.

Differential Testing is also closely associated with regression testing. Regression tests exercise parts of a program to expose bugs introduced due to recent changes. A major challenge in regression testing is test selection which tries to select test cases that are impacted by the modifications. This problem motivated a large amount of research in test selection [5], [14], [18]. They investigated different test selection techniques using cost benefit analysis (testing cost vs. fault detection). Alternative to test selection is test generation which becomes more complicated as the size and complexity of codebase grows [17]. On the other hand random test generation techniques like Randoop [11] do not guarantee high code coverage.

Diffut by Xie et al. performs diff testing on object oriented programs by giving them same input and then compares the output using an automated tree based differ [19]. However, this solution is only applicable to certain domains such as OOP based output. DiffGen [16] tackles this problem by modifying a program so that execution of modified parts of the program can guarantee the detection of changed behavior. BERT [7] performs automated regression testing in a three-step process similar to diff testing. However, BERT analyzes changes in the program and redesigns the test suite to expose bugs in the modified program. The redesigned test suite tests both original and modified version of the program to analyze the behavioral differences including runtime metrics. BERT is an advance tool for diff testing but suffer from flakiness. Chianti uses differential testing to perform test selection [13]. It uses output(diff) of a diff test to select tests whose behaviors are impacted by the applied changes. Such techniques assume that a test suite is available. However, unavailability of a good amount of test cases mitigates the need for test selection.

Flaky tests are ineffective because of inconsistent outcome. Luo et al. empirically analyzed common root causes of flaky tests [9]. According to their investigation, concurrency, test order dependency, resource leak, network, time, I/O, floating point operations, un-ordered collection, etc., are the common causes of flaky tests. Their study also provides suggestions to eliminate flakiness in tests. We believe users of diff test can benefit from the findings of this study to remove flakiness and noise in their test outcome.

In terms of surveys about software testing techniques and practices, we struggled to find related work. Few research works empirically evaluated the effectiveness of new testing methodologies using a benchmark suite [5], [12]. Juristo et al. empirically studied a wide variety of tests over 25 years [8]. They classified testing techniques into families and study the maturity in each family. Regarding regression testing, they did not find any lab study that can provide the perception of the testing techniques.

VII. Conclusion

We present the first empirical study on the usage, benefits, and limitations of differential testing in practice. Our study includes user surveys, interviews, and log analysis. We identified that diff tests suffers from the traditional problem of test data selection and its effectiveness is restricted by a high quality differ module. Diff test is usually used as a sanity check for an entire system and it simulates real production environment which increases confidence in system's quality.

To encourage future research in diff testing, we propose a set of impactful research problems. We believe that optimized data differencing techniques can improve the efficiency and effectiveness of diff testing. Sampling is still an open research problem which can help select test data that can lead to high code coverage. We observe that multiple testing techniques are used simultaneously to improve software quality. A study on the most effective combination of testing techniques can really help practitioners in choosing the best combination of testing methodologies to achieve testing goals.

Acknowledgment

We thank the anonymous reviewers for their comments. We would also like to thank Rob Siemborski and Dejan Desovski at Google for reviewing this paper and providing kind help.

References

[1] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," in *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006, pp. 205–218.

[2] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google&rsquo;s globally distributed database," *ACM Trans. Comput. Syst.*, vol. 31, no. 3, pp. 8:1–8:22, Aug. 2013.

[3] R. B. Evans and A. Savoia, "Differential testing: A new approach to change detection," in *The 6th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers*, ser. ESEC-FSE companion '07. New York, NY, USA: ACM, 2007, pp. 549–552.

[4] I. Google, "Protocol buffers," http://code.google.com/apis/protocolbuffers/.

[5] T. L. Graves, M. J. Harrold, J.-M. Kim, A. Porter, and G. Rothermel, "An empirical study of regression test selection techniques," *ACM Trans. Softw. Eng. Methodol.*, vol. 10, no. 2, pp. 184–208, Apr. 2001.

[6] C. Jaspan, M. Jorde, A. Knight, C. Sadowski, E. K. Smith, C. Winter, and E. Murphy-Hill, "Advantages and disadvantages of a monolithic repository: A case study at google," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '18. New York, NY, USA: ACM, 2018, pp. 225–234.

[7] W. Jin, A. Orso, and T. Xie, "Automated behavioral regression testing," in *2010 Third International Conference on Software Testing, Verification and Validation*, April 2010, pp. 137–146.

[8] N. Juristo, A. M. Moreno, and S. Vegas, "Reviewing 25 years of testing technique experiments," *Empirical Software Engineering*, vol. 9, no. 1, pp. 7–44, Mar 2004.

[9] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov, "An empirical analysis of flaky tests," in *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014. New York, NY, USA: ACM, 2014, pp. 643–653.

[10] W. M. McKeeman, "Differential testing for software," *DIGITAL TECHNICAL JOURNAL*, vol. 10, no. 1, pp. 100–107, 1998.

[11] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball, "Feedback-directed random test generation," in *ICSE 2007, Proceedings of the 29th International Conference on Software Engineering*, Minneapolis, MN, USA, May 2007, pp. 75–84.

[12] X. Qu, M. B. Cohen, and G. Rothermel, "Configuration-aware regression testing: An empirical study of sampling and prioritization," in *Proceedings of the 2008 International Symposium on Software Testing and Analysis*, ser. ISSTA '08. New York, NY, USA: ACM, 2008, pp. 75–86.

[13] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley, "Chianti: A tool for change impact analysis of java programs," in *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, ser. OOPSLA '04. New York, NY, USA: ACM, 2004, pp. 432–448.

[14] G. Rothermel and M. J. Harrold, "A safe, efficient regression test selection technique," *ACM Trans. Softw. Eng. Methodol.*, vol. 6, no. 2, pp. 173–210, Apr. 1997.

[15] C. Sutton, T. Hobson, J. Geddes, and R. Caruana, "Data diff: Interpretable, executable summaries of changes in distributions for data wrangling," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &#38; Data Mining*, ser. KDD '18. New York, NY, USA: ACM, 2018, pp. 2279–2288.

[16] K. Taneja and T. Xie, "Diffgen: Automated regression unit-test generation," in *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, Sept 2008, pp. 407–410.

[17] W. Visser, C. S. Păsăreanu, and S. Khurshid, "Test input generation with java pathfinder," in *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA '04. New York, NY, USA: ACM, 2004, pp. 97–107.

[18] W. E. Wong, J. R. Horgan, S. London, and H. A. Bellcore, "A study of effective regression testing in practice," in *Proceedings of the Eighth International Symposium on Software Reliability Engineering*, ser. ISSRE '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 264–274.

[19] T. Xie, K. Taneja, S. Kale, and D. Marinov, "Towards a framework for differential unit testing of object-oriented programs," in *Automation of Software Test, 2007. AST'07.* IEEE, 2007, pp. 5–11.