2018 25th Asia-Pacific Software Engineering Conference (APSEC)

On the Use of Automated Log Clustering to Support Effort Reduction in Continuous Engineering

Carl Martin Rosenberg
Simula Research Laboratory
Oslo, Norway
Email: cmr@simula.no

Leon Moonen
Simula Research Laboratory
Oslo, Norway
Email: leon.moonen@computer.org

***Abstract*—Continuous engineering (CE) practices, such as continuous integration and continuous deployment, have become key to modern software development. They are characterized by short automated build and test cycles that give developers early feedback on potential issues. CE practices help to release software more frequently, and reduces risk by increasing incrementality. However, *effective* use of CE practices in industrial projects requires making sense of the vast amounts of data that results from the repeated build and test cycles.**

The goal of this paper is to investigate to what extent these data can be treated more effectively by automatically grouping logs of runs that failed for the same underlying reasons, and what effort reduction can be achieved. To this end, we replicate and extend earlier work on system log clustering to evaluate its efficacy in the CE context, and to investigate the impact of five alternative log vectorization techniques.

We built a prototype tool that is used to conduct an empirical case study on continuous deployment logs provided by our industrial collaborator. Questions to be answered include: (1) Can we reduce the effort needed to discover all latent issues in a set of failing runs? (2) How to best leverage the contrast between passing and failing runs to increase accuracy? (3) What trade-offs are there between effort reduction and accuracy? We present a quantitative and qualitative analysis of the results of our study. We conclude by evaluating the trade-offs, and give recommendations for applying this approach in practice.

I. Introduction

In order to meet rising demands for frequent releases and high service levels, modern software development sees an increased adoption Continuous Engineering (CE) practices: Continuous Integration (CI) aims at automatically building and (unit) testing software changes multiple times a day, Continuous Delivery (CDy) extends CI with automated acceptance testing and quality checking to ensure that a product is ready for deployment, Continuous Deployment (CDt) adds automatic deployment to production-like hardware and deployment testing, and Continuous Release (CR) further adds automatically releasing the new software to the customers.[1] When there is no need to distinguish, we will refer to these practices as CE.

Each of the practices focuses on creating short automated cycles that give developers early feedback on potential issues and reduce risk by increasing incrementality. The cycles can be triggered by new commits to the versioning system, via periodic scripts, or simply on demand by a developer.

[1] There is an ongoing debate on the definitions and boundaries of these practices, but we will use this incremental CI ⊂ CDy ⊂ CDt ⊂ CR definition.

A frequently reported challenge for the adoption of CE is the need for systematic analysis of the abundance of data resulting from the automated build, test, and deployment processes [1–6]. Without CE, a developer would manually build, test or deploy the system, observe the results and immediately react to them. With CE, an automated cycle is started that stores its results in a log, the developer switches to a new tasks, and the log is inspected later. This increases productivity when the cycles are time-consuming. However, it can also lead to the accumulation of vast amounts of unprocessed results, especially when combined with extensive automatic testing.

Goal: We investigate how the analysis of accumulated CE logs can be automatically supported, and what effort reduction can be achieved. More specifically, we aim to automatically group logs of CE runs that failed for the same underlying reasons. The idea being that automated log clustering enables a more systematic, coordinated approach where one investigates a representative for a group of failed runs, instead of having to investigate all individual results. Moreover, after addressing the issue that caused failure for a representative of a cluster, one can reasonably expect that the issues of logs that failed for corresponding reasons have been accounted for (which will be checked by following runs).

Contributions: We replicate and extend earlier work by Lin et al. [7] that proposed automatic clustering for system logs. Out main contributions are the following: (1) We conduct a replication study to investigate the efficacy of an approach developed for identifying problems in system logs in the context of analyzing CE logs, and the effort reduction that can be achieved. (2) We extend the original study in two ways: (2a) we investigate an alternative technique to exploit the *frequency* with which events occur in passing and failing logs, and (2b) we investigate two alternative techniques to exploit the *contrast* between events occurring only in failing logs and events that also occur in passing logs. (3) We empirically evaluate the impact of the 6 resulting configurations on the quality of clustering CDt logs provided by our industrial collaborator. (4) We analyze and discuss how the various pipelines affect the effort reduction that can be achieved by clustering CDt logs.

Overview: The remainder of this paper is organized as follows: Section II discusses Lin's approach, and Section III presents the variations that we investigate. Section V describes

978-1-7281-1970-0/18/$31.00 ©2018 IEEE
DOI 10.1109/APSEC.2018.00032

```
2018-01-16 11:32:57.111 CET: [test-setup] {INFO} Check that system are in the correct default initial state
2018-01-16 11:32:57.113 CET: [test-setup] {INFO} * System check not supported for ce-host, u'vm-ce-host5.qa'
2018-01-16 11:32:58.230 CET: [unfit] {INFO} Test Success get_keys
2018-01-16 11:32:59.108 CET: [unfit] {INFO} Test Success get_platform_sanity
2018-01-16 11:32:59.250 CET: [unfit] {INFO} Test Success get_input_status
2018-01-16 11:32:59.317 CET: [unfit] {INFO} Test Success get_camera_lid_state
2018-01-16 11:32:59.794 CET: [unfit] {INFO} Test Success get_keys
2018-01-16 11:33:00.791 CET: [unfit] {INFO} Test Success get_input_status
2018-01-16 11:33:00.908 CET: [unfit] {INFO} Test Success get_camera_lid_state
2018-01-16 11:33:06.096 CET: [fixture.py] {INFO} Dial reference from testtarget on h323 with callrate 3000
```

Listing 1. An excerpt from one of the CDt logs of our industrial partner.

our experimental design, whose results are presented and discussed in Section VI. Section VIII presents related work, and Section IX provides some concluding remarks.

II. Background

As in our other work that investigates the impact of dimensionality reduction on log clustering [8], the baseline for our investigation is LogCluster by Lin et al. [7]. We share LogCluster's choice of clustering algorithm (Hierarchical Agglomerative Clustering–HAC) and distance metric (cosine distance), but investigate variations with respect to the treatment of logs before clustering. This section explains how logs are treated in LogCluster and Section III introduces the variations on LogCluster investigated in this paper.

In essence, LogCluster adopts and adapts the common *bag-of-words* approach to document clustering [9] in which every document is represented as a vector of word frequencies: If there are n unique words in the set of documents to be clustered, every document will be represented as an n-dimensional vector v where each element in the vector tracks how often a specific word occurs in the document. However, LogCluster differs from the standard bag-of-words approach in two important ways: It uses *events* rather than words as the fundamental unit of abstraction, a *bag-of-events* model if you will, and it applies a weighing scheme that attempts to give higher significance to events that only occur in failing logs.

Deriving a bag-of-events representation: There are three steps to deriving a bag-of-events representation. The first step is to assume that the logs being clustered can be interpreted as a sequence of events, and that each event in the log can be delineated by a machine-recognizable pattern. As a running example, we will use an excerpt from of of the CDt logs of our industrial collaborator, shown in Listing 1. In the log excerpt the event delineation pattern is a newline followed by a date, a time-stamp and a time-zone indicator. The second step is to perform *log abstraction* by removing all runtime-specific information that creates artificial differences between events, especially dates and timestamps. For example, we would want the occurrences of `Test Success get_keys` in Listing 1 to be treated as the same event occurring two times rather than as two unique events. While the original LogCluster pipeline uses a log abstraction mechanism by Fu et al. [10], we make use of a log abstraction tool developed by Cisco Systems Norway. Finally, the third and last step is to automatically split each log into events, determine the set of unique events occurring in the logs, and representing each log as a vector of event frequencies.

Setting event weights: It is generally assumed in Information Retrieval that rare words are more informative than common words. To account for this fact, the *inverse document frequency scheme* gives higher weights to rarely occurring words, and proportionally lower weight to common words [9]. LogCluster employs an inverse document frequency scheme to assign higher priority to rarely occurring events such that the weight $w_f(e)$ of an event e is given by: $w_f(e) = S(\log \frac{N}{n_e})$ where N is the number of logs to be clustered, n_e is the number of logs where the event e appears, and S is the Sigmoid function $1/(1+\exp[-x])$ which normalizes the vector by ensuring that all values are between 0 and 1.

Furthermore, LogCluster is concerned with effectively clustering production logs by using knowledge of corresponding laboratory logs: It posits that events that are *only* observed in production logs are extra informative, and that the event weight should reflect this. Lin et al. refer to this technique as *contrast-based event weighing*. This is implemented in LogCluster by computing the set of production-only events ΔS, and then letting the final weight of the event be set by: $w(e) = 0.5 * w_{con}(e) + 0.5 * w_f(e)$, where $w_f(e)$ is the classic idf weight computed above and $w_{con}(e)$ is the *contrast weight* so that $w_{con}(e) = 1$ if $e \in \Delta S$ and $w_{con}(e) = 0$ otherwise.

We adapt this method for our use-case by leveraging the contrast between passing and failing logs, letting ΔS denote the set of events unique to failing logs. We investigate alternative ways of incorporating the contrast between failing and passing logs, and will refer to Lin et al.'s concrete strategy as *proportional contrast* in the remainder of this paper.

Performing the clustering and selecting representatives: The clustering is performed by applying Hierarchical Agglomerative Clustering (HAC) using *cosine distance* as distance metric. The clustering algorithm stops merging subclusters when the distance between two subclusters exceed a threshold θ, which LogCluster sets to 0.5. When the clusters have been computed, the log in each cluster with the smallest average cosine distance to the other logs in its cluster is extracted as a representative.

III. Variations on LogCluster

Having described the main tenets of LogCluster, we now present and motivate the additional variations that we investigate in this paper. Figure 1 shows a high level overview of the LogCluster pipeline and our variations.

Alternative event frequency weighting scheme: We investigate whether an improvement can be made by employing a *tf-idf* scheme for weighting event frequency vectors. This choice

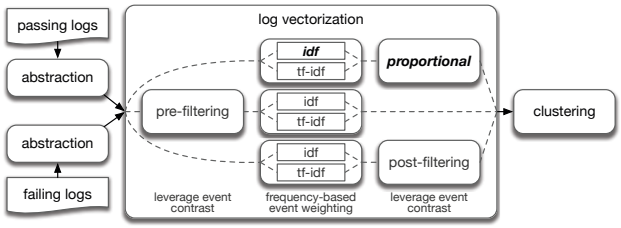

Fig. 1. Overview of the LogCluster pipeline (top flow, in bold italic) together with the additional variations investigated in this paper.

may be more appropriate for CE logs which contain lower frequencies of repeated events per log than typical system event logs, a fact which may be successfully exploited by using *tf-idf* instead of *idf* for event frequency weighting. Tf-idf uses the number of times an event occurs in a log instead of just whether it occurs or not (as used by idf). The tf-idf based event frequency weight $w_f(e)$ is determined by $w_f(e) = tf(e,d) \times (log(\frac{N+1}{n_e+1}) + 1)$, where N is the number of logs being clustered, n_e is the number of logs where the event e appears and $tf(e,d)$ denotes the number of times event e occurs in document d. We add one to the numerator and denominator in the second term to prevent divisions by zero, and add one to the second term to ensure that events occurring in all documents (i.e. for which $n_e = N$) also get a nonzero weight. After computing this score for every event in the log, all the scores are normalized with the L_2 (Euclidean) norm to ensure that all resulting values fall in the interval $[0, 1]$.

Alternative techniques to leverage event contrast: LogCluster's *proportional contrast* leverages the contrast between events that only occur in failing logs and events that occur in both failing and passing logs. Our work investigates two alternative techniques to leverage event contrast that we refer to as *pre-filtering* and *post-filtering*.

Pre-filtering: This technique is based on the idea that the only events that matter for clustering accuracy are the events that only occur in failing logs (*fail-only* events). Thus, pre-filtering removes all events that occur in both failing and passing logs (*co-occurring* events) from the failing logs before feeding them to the event frequency weighting step.

Post-filtering: This technique is based on the idea that event frequency can be used to leverage the contrast between events in failing and passing logs: when the number of passing logs exceeds the number of failing logs (which is not an unreasonable expectation), then inverse document frequency will automatically highlight fail-only events and downplay co-occurring events. Thus, in post-filtering, both the passing and failing logs are fed into the event frequency weighting step and afterward the passing logs are discarded, so that only the failing logs gets clustered.

IV. Research Questions

This paper is concerned with answering the following three research questions:

RQ1 Can we reduce the effort needed to discover all latent issues in a set of failing runs?

RQ2 How to best leverage the contrast between passing and failing runs to increase accuracy?

RQ3 What trade-offs are there between effort reduction and accuracy?

V. Experimental design

We answer our research questions by measuring and analyzing the performance of LogCluster and the variations discussed in Section III. This section presents the dataset, measures and statistical procedures used.

A. Datasets

For our empirical evaluation, we use the same dataset as was used in our investigation of the impact of dimensionality reduction on log clustering [8]. This dataset was provided by our industrial partner Cisco Norway, a worldwide leader in the production of software-intensive embedded systems in the networking domain. The logs provided were extracted from recent development activity on their main software product line for professional video conferencing systems. Cisco uses continuous deployment: changes committed to the VCS are automatically integrated, built, and deployed to variants of their hardware for comprehensive interoperability testing.

Each dataset used in this investigation concerns a specific integration test, and consists of logs for both failing and passing runs of the given test. The logs are loosely structured execution outputs that capture the process of building, deploying and testing various scenarios, not unlike the output of running a make command, as can be seen in Listing 1. The eighteen datasets used in this investigation are summarized in Table I.

The initial dataset was filtered so that every failing run is associated with one (and only one) known issue. In earlier work, Cisco Norway handcrafted regular expressions to identify whether a log concerns a known issue. We use these expressions to establish a *ground truth* for how the logs should be clustered together so that each cluster only contains logs

TABLE I
Main characteristics of the input datasets used.

dataset	failing logs	passing logs	distinct events	fail-only events	co-occurring events	ground-truth clusters
1	45	4189	220	57	107	7
2	102	4182	308	90	102	13
3	103	1315	6193	271	579	5
4	8	4277	78	2	68	1
5	41	297	138	19	99	3
6	23	2421	1176	109	243	9
7	50	1591	90	15	56	5
8	78	3213	1488	404	201	11
9	52	461	227	57	120	9
10	342	3152	636	105	96	30
11	15	3816	67	1	22	1
12	74	2811	134	73	60	8
13	106	572	253	43	138	12
14	60	613	220	41	121	11
15	140	3247	295	98	106	20
16	132	3238	321	101	112	23
17	36	530	283	68	133	10
18	29	344	249	48	123	8

concerning the same issue. This allows us to evaluate the accuracy of clusterings proposed by the various pipelines.

B. Quality Measures

Adjusted Mutual Information: We use *Adjusted Mutual Information* (AMI) [11] to measure how well each configuration clusters the datasets in accordance with the ground truth. While Lin et al. [7] use *Normalized Mutual Information* [9] for this purpose, NMI has a systematic bias in favor of clustering algorithms that group data into many small clusters, as those are more likely to have many agreements solely due to chance [11]. The AMI measure corrects for this bias [11].

An AMI score can at most be 1, which indicates a perfect correspondence between the proposed clustering and the ground truth. An AMI score near 0, on the other hand, indicates that the proposed clustering performs as one would expect from a solution based on random guessing.[2]

Homogeneity and Completeness: Homogeneity (H) and Completeness (C) represent two competing clustering quality concerns, and are defined in relation to the *V-measure*, which is the weighted harmonic mean of the two [12]. Homogeneity measures the extent to which members of a proposed cluster come from the same ground-truth cluster, while Completeness measures the extent to which all members of a given ground-truth cluster occur in the same proposed cluster. A clustering algorithm can trivially obtain a perfect Homogeneity score by putting each log into a one-membered cluster (resulting in one cluster per log), but such a solution would obtain a very low Completeness score. On the other hand, a clustering algorithm can trivially obtain a perfect Completeness score by creating only one cluster and placing every log in that cluster, but such a solution would obtain a very low Homogeneity score. Thus, a well-performing clustering algorithm must strike a good balance between both Homogeneity and Completeness.

Homogeneity and Completeness scores range from 0 to 1, where 1 is a perfect score and 0 is the worst possible score.

Effort Reduction: Raw, ideal and scaled: In a perfect world, our clustering algorithm would only group together logs that concern the exact same issue. A user would cluster n failing logs, extract r representatives–one representative per cluster– and manually inspect r rather than n logs, where r is typically much smaller than n. In that case, the proportional reduction in required effort, or simply *effort reduction*, would be given by $\text{ER} = 1 - \frac{\text{r}}{n}$.

However, in an imperfect world, a clustering algorithm might create too few or too many clusters. To get a sense of how good the observed effort reduction is, it helps to know what effort reduction a perfectly accurate algorithm would achieve. We refer to this as the *ideal effort reduction*. Recall that the algorithm produces one representative per cluster. Therefore, the ideal effort reduction (IER) is given by $\text{IER} = 1 - \frac{\text{distinct ground-truth clusters}}{n}$.

[2] This has an important implication for interpreting our results: An AMI score of 0.5 does *not* correspond to random guessing, as opposed to many other accuracy measures (e.g., the F1 score).

However, since ER and IER only reflect the *number* of clusters rather than the cluster contents, we might have clustering outcomes where ER = IER but where the contents of the clusters completely deviate from the ground-truth. We therefore define *scaled effort reduction* (SER) as $\text{SER} = \text{ER} * \text{H}$. Recall that Homogeneity measures to what extent members of a cluster come from the same ground-truth cluster. Ideally, we want every cluster to contain members from only one ground-truth cluster, so that the cluster representatives genuinely represent all the logs in the cluster. The Homogeneity score measures to what extent this is the case. By scaling the effort reduction with Homogeneity we get a sense of the quality of the effort reduction achieved as well. In a perfect world we would obtain $\text{SER} = \text{ER} * \text{H} = \text{IER} * 1 = \text{IER}$.

C. Statistical Procedures

To answer RQ2 and RQ3, we are interested in determining whether there are significant differences between the variations we consider. In order to do so, we first make use of the non-parametric Friedman test [13] to establish whether there is at least one significant difference between the variants.[3] If the Friedman test produces a p-value below or at significance level $\alpha = 0.05$, we reject the null hypothesis of no significant difference between the variants and proceed with a post-hoc test to pinpoint where the differences lie.

As a post-hoc test, we compare each pair of variants using two-sided, paired Wilcoxon Signed Rank Tests [15] with the Pratt-correction [16] to account for ties as conservatively as possible.[4] Since the post-hoc tests cause multiple comparisons, we use Holm's procedure [18] to adjust our significance level α to control the family-wise error rate.

Once we have established which variants are statistically similar or different, we visualize the pairwise comparisons with a compact letter display generated by Piepho's algorithm [19]. If two treatments do not differ in a statistically significant way, they are given the same letter.

We also record the the Vargha-Delaney A_{12} and A_{21} effect size measures [20]: When comparing a against b, A_{ab} approximates the *stochastic superiority* of a over b. For the same comparison, $A_{ba} = 1 - A_{ab}$ measures the stochastic superiority of b over a. By convention, a VDA of .56 is considered *small*, a VDA of .64 is considered *medium* and VDA of .71 or above is considered *large* [20].

For running these statistical tests, we made use of the `stats.wilcoxon` and `stats.friedmanchisquare` procedures from SciPy [21]. The plots are made with ggplot2 [22] in R [23]. The compact letter displays are generated with `multcompView` [24] in R.

[3] We refer to Demšar [14] for a treatment on why the non-parametric Friedman test is to be preferred over parametric approaches such as ANOVA when comparing algorithms.

[4] We refer to Benavoli et al. [17] for an argument for using the Wilcoxon Signed-Rank test as post-hoc procedure after the Friedman test.

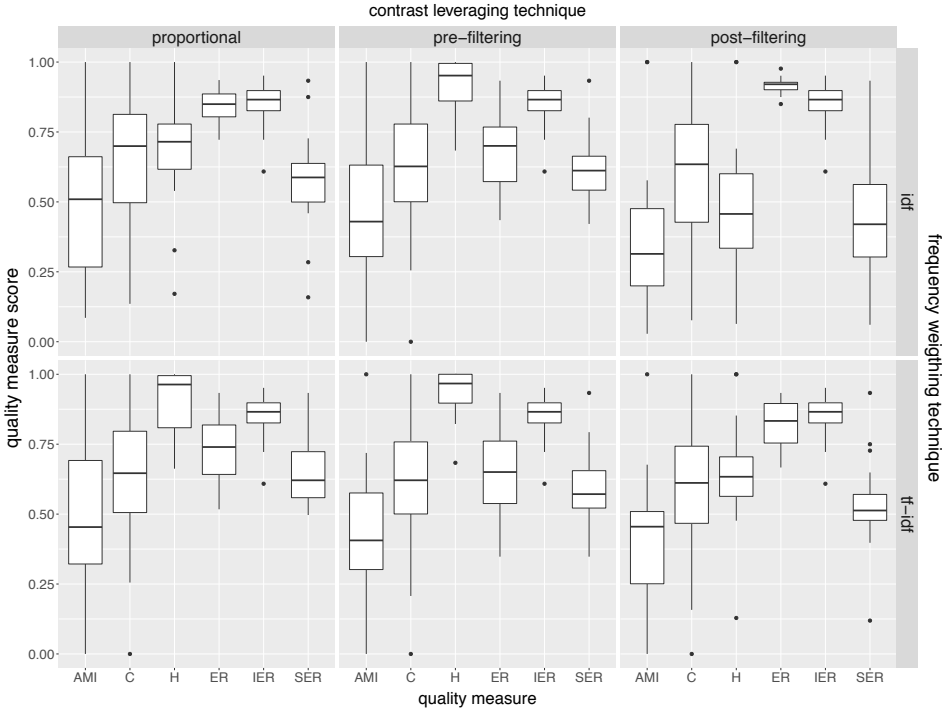

Fig. 2. Aggregated scores for each variant

VI. Results and Discussion

A. Overview

Table II and Figures 2 and 3 summarize our results, and we refer to these tables and figures extensively in the following discussion. Table II summarizes the median performance of all variants on AMI, Homogeneity, Effort Reduction, and Scaled Effort Reduction. The spread of these values is illustrated with boxplots in Figure 2. Moreover, Figure 3 shows the performance of the variants *per dataset*, and highlights how well the effort reduction scores fare against the ideal effort reduction score.

B. Baseline performance

Since LogCluster by Lin et al. [7] serves as a baseline for our study, we first concentrate on the exact performance of this algorithm on our dataset. Recall that LogCluster implies using the *proportional contrast* in conjunction with the *idf* weighing scheme discussed in Section II.

TABLE II
Median performance of all variants on AMI, Homogeneity, Effort Reduction, and Scaled Effort Reduction

freq. weighting	contrast leverage	median AMI	median ER	median SER	median H
idf	proportional	0.5094	0.8495	0.5874	0.7150
tf-idf	proportional	0.4537	0.7399	0.6209	0.9634
idf	pre-filter	0.4295	0.7002	0.6119	0.9516
tf-idf	pre-filter	0.4060	0.6504	0.5715	0.9670
idf	post-filter	0.3141	0.9206	0.4200	0.4569
tf-idf	post-filter	0.4551	0.8332	0.5129	0.6336

The exact performance of LogCluster is shown in Table III: For ER and scaled ER, we report the difference from IER in parentheses. A negative result indicates that the effort reduction is lower than ideal. A positive effort reduction, moreover, means that the raw effort reduction exceeds the ideal effort reduction, which implies that the clustering algorithm has produced fewer clusters than the ground truth. The Clusters-column shows how many clusters LogCluster produced, with the number of clusters an ideal algorithm would produce

TABLE III
Results on LogCluster's exact performance

Dataset	AMI	H	C	ER	SER	Clusters
1	0.383	0.775	0.537	0.733 (-0.111)	0.568 (-0.276)	12 (7)
2	0.673	0.853	0.747	0.853 (-0.02)	0.727 (-0.145)	15 (13)
3	0.201	0.676	0.251	0.883 (-0.068)	0.597 (-0.354)	12 (5)
4	1.0	1.0	1.0	0.875 (=)	0.875 (=)	1 (=)
5	0.085	0.171	0.135	0.927 (=)	0.159 (-0.768)	3 (=)
6	0.211	0.327	1.0	0.87 (+0.261)	0.284 (-0.324)	3 (9)
7	0.676	0.705	1.0	0.92 (+0.02)	0.649 (-0.251)	4 (5)
8	0.341	0.722	0.494	0.808 (-0.051)	0.583 (-0.276)	15 (11)
9	0.623	0.708	0.815	0.846 (+0.019)	0.599 (-0.228)	8 (9)
10	0.569	0.645	0.677	0.936 (+0.023)	0.604 (-0.309)	22 (30)
11	1.0	1.0	1.0	0.933 (=)	0.933 (=)	1 (=)
12	0.453	0.734	0.574	0.797 (-0.095)	0.586 (-0.306)	15 (8)
13	0.2	0.539	0.317	0.887 (=)	0.478 (-0.409)	12 (=)
14	0.263	0.562	0.445	0.817 (=)	0.459 (-0.357)	11 (=)
15	0.734	0.855	0.808	0.836 (-0.021)	0.714 (-0.143)	23 (20)
16	0.565	0.734	0.722	0.803 (-0.023)	0.589 (-0.236)	26 (23)
17	0.628	0.78	0.796	0.722 (=)	0.563 (-0.159)	10 (=)
18	0.281	0.607	0.506	0.759 (+0.034)	0.461 (-0.264)	7 (8)
Median	0.509	0.715	0.700	0.850	0.587	-
Mean	0.494	0.689	0.657	0.845	0.579	-

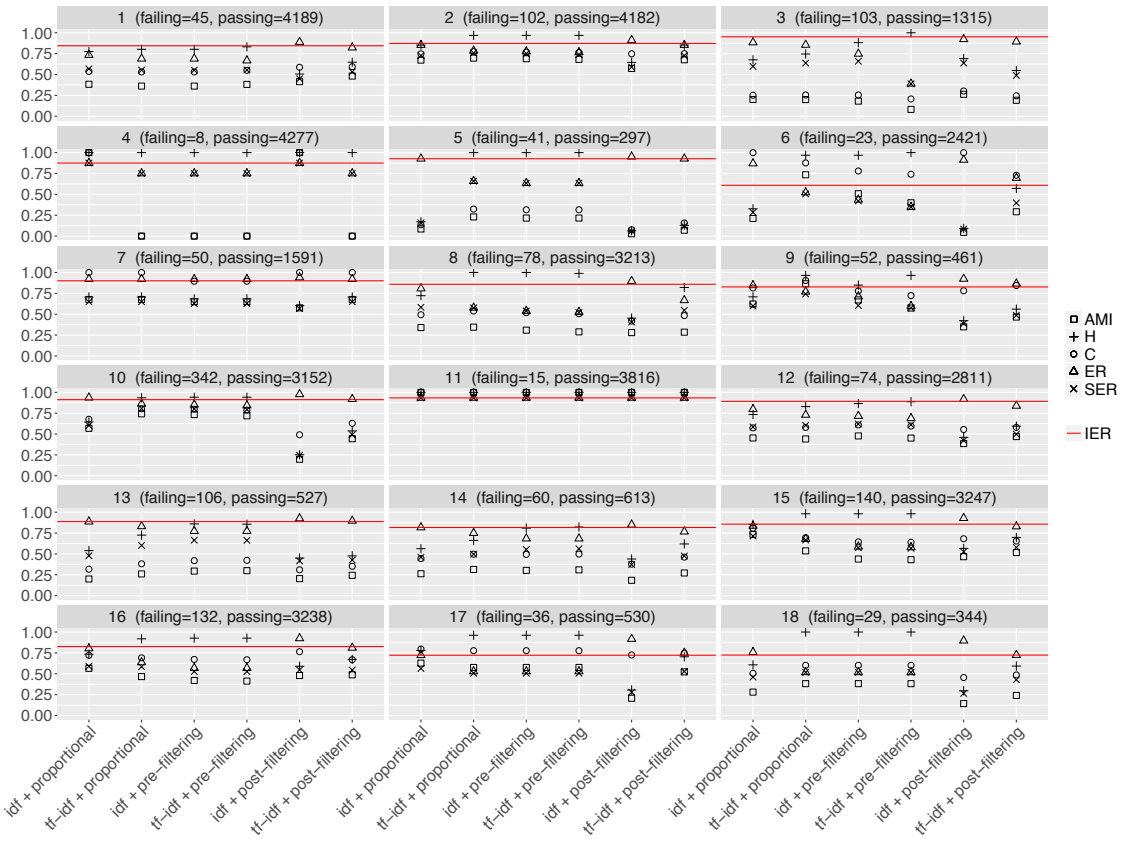

Fig. 3. Per-test performance

indicated in parentheses.

Overall, the raw effort reduction is roughly 5 to 25 percent below the ideal rate. This can also be seen visually in Figure 3, where the ER mark for LogCluster is close-to, but not on, the red line indicating the IER for that test. On datasets 6,7,9, 10 and 18 the ER exceeds IER, and for these cases the Clusters column shows us that LogCluster has produced fewer clusters than the ideal solution would.

Datasets 5, 13, 14 and 17 demonstrates the importance of having a Homogeneity-scaled effort reduction measure: While LogCluster produces the same *number* of clusters as the ideal solution, these clusterings range from the severely (5) to somewhat (17) inaccurate, as indicated by the AMI and H scores. The scaled Effort Reduction score incorporates this nicely by reporting a very low score for pathological cases (dataset 5) and a moderately penalized score where the number of clusters are the same as in the ground truth, but where the cluster contents differ somewhat (dataset 17).

While LogCluster achieves a perfect ER and SER score on datasets 4 and 11, these are the two smallest datasets and only require the algorithm to lump all logs into one cluster to achieve a perfect score. This does not imply that the dataset is trivial, however, since the algorithm must still identify that the correct action is to create a single cluster. As is shown in Figure 3 not all variants do this correctly for dataset 4.

C. Performance of all variants

Figure 3 shows the performance of each variant on each individual test. Overall, we see that the ideal effort reduction IER is typically above .75 for all datasets, except dataset 6 where it is .60 and datasets 17 and 18 where it is .72. The variants score an ER close to the IER on datasets 3,4, 7, 10, 11, 13 and 14. On datasets 1,2,8,9,15,16,17 and 18, however, the majority of variants score an ER systematically below the IER. The post-filtering variants achieve a higher ER on datasets 6 and 12 than the other variants, but unfortunately also attain very low AMI and SER on those same datasets.

In terms of Homogeneity, Proportional contrast with tf-idf and the two pre-filtering variants attain higher Homogeneity scores than other variants on all datasets except 1,4,7,11, and 12. However, we see that these variants score a proportionally lower ER score than other variants on dataset 5,6,8,15,16,17 and 18, suggesting that the variants have an inherent bias for preferring small homogeneous clusters.

For about half of the datasets, we see that AMI and SER scores are similar. On datasets 1,3,4,5,8,13 and 14, however, AMI and SER visibly diverge. An interesting example is dataset 4, where the correct answer is to put all logs in a single cluster. Proportional contrast with tf-idf and the pre-filtering variants create two rather than one cluster, obtaining perfect Homogeneity and a high effort reduction but gets heavily

TABLE IV
SUMMARY OF AMI COMPARISONS

a	b	p-value	wilcox	VDA_{12}	VDA_{21}
idf + prop	idf + pre	0.7439	78	0.5216	0.4784
idf + prop	idf + post	**0.0018**	14	*0.6420*	0.3580
idf + prop	tf-idf + prop	0.4204	67	0.4799	0.5201
idf + prop	tf-idf + pre	0.6165	74	0.5370	0.4630
idf + prop	tf-idf + post	0.1024	48	0.5849	0.4151
idf + pre	idf + post	0.0311	36	0.6296	0.3704
idf + pre	tf-idf + prop	0.0096	26	0.4444	0.5556
idf + pre	tf-idf + pre	0.0123	28	0.5278	0.4722
idf + pre	tf-idf + post	0.0936	47	0.5617	0.4383
idf + post	tf-idf + prop	0.0139	29	0.3611	0.6389
idf + post	tf-idf + pre	0.0582	42	0.3920	0.6080
idf + post	tf-idf + post	0.0096	26	0.4198	0.5802
tf-idf + prop	tf-idf + pre	0.0108	27	0.5648	0.4352
tf-idf + prop	tf-idf + post	0.0156	30	0.5941	0.4059
tf-idf + pre	tf-idf + post	0.2485	59	0.5370	0.4630

p-values that are significant after Holm correction in bold, medium effect size in italics, large effect size in bold

TABLE VI
SUMMARY OF ER COMPARISONS

a	b	p-value	wilcox	VDA_{12}	VDA_{21}
idf + prop	idf + pre	**0.0002**	0	**0.8457**	0.1543
idf + prop	idf + post	**0.0002**	0	0.1821	**0.8179**
idf + prop	tf-idf + prop	**0.0002**	0	**0.7870**	0.2130
idf + prop	tf-idf + pre	**0.0002**	0	**0.8549**	0.1451
idf + prop	tf-idf + post	0.4997	70	0.5556	0.4444
idf + pre	idf + post	**0.0002**	0	0.0756	**0.9244**
idf + pre	tf-idf + prop	**0.0002**	0	0.4275	0.5725
idf + pre	tf-idf + pre	**0.0002**	0	0.5694	0.4306
idf + pre	tf-idf + post	**0.0002**	0	0.2099	**0.7901**
idf + post	tf-idf + prop	**0.0002**	0	**0.9213**	0.0787
idf + post	tf-idf + pre	**0.0002**	0	**0.9275**	0.0725
idf + post	tf-idf + post	**0.0002**	0	**0.8380**	0.1620
tf-idf + prop	tf-idf + pre	**0.0002**	0	0.6235	0.3765
tf-idf + prop	tf-idf + post	**0.0002**	0	0.2747	**0.7253**
tf-idf + pre	tf-idf + post	**0.0002**	0	0.1867	**0.8133**

p-values that are significant after Holm correction in bold, medium effect size in italics, large effect size in bold

penalized in terms of AMI.

These observations are substantiated in the boxplots in Figure 2: Proportional contrast with tf-idf and the pre-filtering variants attain very high Homogeneity scores, while the post-filtering variants attain very high raw ER scores.

D. Statistical Analysis

As discussed in the experimental design (Section V), we consider a total of 6 different vectorization strategies. With 18 datasets, we thus have 6 different *treatments* being measured on 18 different *blocks*.

We run three statistical tests with corresponding post-hoc procedures. Each test targets a specific quality measure. The quality measures considered are AMI, ER and SER.

AMI: The Friedman test yields a test statistic of 18.6806 at a p-value of 0.0022. We thus reject the null hypothesis of equal performance for all configurations at significance level $\alpha = 0.05$, and continue with the pairwise post-hoc tests.

The post hoc-tests are summarized in Table IV. Our investigation found only one significant difference between the variants: *idf and proportional contrast* was found to perform significantly better than *idf and post-filtering*, with a Vargha-Delaney effect size of 0.6420. According to the convention of interpreting Vargha-Delaney scores this is interpreted as a *medium* effect size. The mean, median, max and min AMI scores for each variant are listed in Table V.

ER: Our Friedman test results in a test statistic of 76.9308 at a p-value of $3.6769e{-}15$, leading us to reject the null hypothesis of equal performance for all configurations at significance level $\alpha = 0.05$ and proceed with the post-hoc tests. The post-hoc tests are summarized in Table VI, and the group memberships and summarizing statistics are shown in Table VII. As seen in Table VI, all except one comparison yielded a statistically significant difference. Post-filtering with idf obtains the highest median and mean ER scores and is significantly different from all other variants in this regard. Proportional contrast with idf and post-filtering with tf-idf share second place.

Scaled ER: Once again we run the Friedman test, resulting in a test statistic of 33.1469 at a p-value of $3.5189e-06$, and so we reject the null hypothesis of equal performance for all configurations at significance level $\alpha = 0.05$. The post-hoc tests are summarized in Table VIII, with group memberships and key summary statistics listed in Table IX. Only six out of fifteen comparisons found a statistically significant difference. When corroborated with the summary statistics in Table IX, we see that the significant comparisons (and correspondingly large effect sizes) concern comparisons where one of the low-performing post-filtering variants is compared against one of the better performing variants. Indeed, the clearest signal from this test is that post-filtering should not be preferred when optimizing for SER.

E. Response to research questions

RQ1: *Can we reduce the effort needed to discover all latent issues in a set of failing runs?* – We see that the LogCluster baseline achieves a considerable *raw* median effort

TABLE V
SUMMARY OF AMI PERFORMANCE PER VARIANT

group	freq. weighting	contrast leverage	median AMI	mean AMI	max AMI	min AMI
a	idf	proportional	0.5094	0.4937	1.0000	0.0854
ab	tf-idf	post-filtering	0.4551	0.4076	1.0000	0.0000
ab	tf-idf	proportional	0.4537	0.4905	1.0000	0.0000
ab	idf	pre-filtering	0.4295	0.4562	1.0000	0.0000
ab	tf-idf	pre-filtering	0.4060	0.4364	1.0000	0.0000
b	idf	post-filtering	0.3141	0.3775	1.0000	0.0285

TABLE VII
SUMMARY OF ER PERFORMANCE PER VARIANT

group	freq. weighting	contrast leverage	median ER	mean ER	max ER	min ER
a	idf	proportional	0.8495	0.8447	0.9357	0.7222
a	tf-idf	post-filtering	0.8332	0.8255	0.9333	0.6667
b	idf	pre-filtering	0.7002	0.6865	0.9333	0.4348
c	idf	post-filtering	0.9206	0.9162	0.9766	0.8500
d	tf-idf	proportional	0.7399	0.7217	0.9333	0.5172
e	tf-idf	pre-filtering	0.6504	0.6504	0.9333	0.3478

TABLE VIII
SUMMARY OF SCALED ER COMPARISONS

a	b	p-value	wilcox	VDA_{12}	VDA_{21}
idf + prop	idf + pre	0.2860	61	0.4306	0.5694
idf + prop	idf + post	**0.0003**	3	**0.7315**	0.2685
idf + prop	tf-idf + prop	0.0936	47	0.3796	0.6204
idf + prop	tf-idf + pre	0.8789	82	0.4892	0.5108
idf + prop	tf-idf + post	**0.0038**	19	0.6327	0.3673
idf + pre	idf + post	**0.0009**	9	**0.7793**	0.2207
idf + pre	tf-idf + prop	0.0854	46	0.4614	0.5386
idf + pre	tf-idf + pre	**0.0050**	21	0.5509	0.4491
idf + pre	tf-idf + post	0.0084	25	**0.7191**	0.2809
idf + post	tf-idf + prop	**0.0006**	7	0.1960	**0.8040**
idf + post	tf-idf + pre	0.0065	23	0.2731	**0.7269**
idf + post	tf-idf + post	0.0108	27	0.3441	*0.6559*
tf-idf + prop	tf-idf + pre	0.0386	38	0.5880	0.4120
tf-idf + prop	tf-idf + post	**0.0004**	4	**0.7577**	0.2423
tf-idf + pre	tf-idf + post	0.1701	54	*0.6605*	0.3395

p-values that are significant after Holm correction in bold, medium effect size in italics, large effect size in bold

TABLE IX
SUMMARY OF SER PERFORMANCE PER VARIANT

group	freq. weighting	contrast leverage	median SER	mean SER	max SER	min SER
ab	idf	proportional	0.5874	0.5794	0.9333	0.1588
ab	tf-idf	proportional	0.6209	0.6416	0.9333	0.4968
a c	idf	pre-filtering	0.6119	0.6243	0.9333	0.4212
b d	tf-idf	pre-filtering	0.5715	0.6011	0.9333	0.3478
cd	tf-idf	post-filtering	0.5129	0.5335	0.9333	0.1192
d	idf	post-filtering	0.4200	0.4481	0.9333	0.0607

reduction score of .8495. However, scaling this number by the Homogeneity of the obtained clusters, we obtain a scaled effort reduction score of .5874. So while the effort reduction can be considerable, the user should not exclusively rely upon this tool when troubleshooting issues, as there is a chance for false positives.

RQ2: *How to best leverage the contrast between passing and failing runs to increase accuracy?* – Our experiments could not determine any significant difference between the considered variations in terms of AMI. However, if one interprets SER as an alternative form of accuracy measure, our tests advice against using the post-filtering strategy and rather preferring the proportional contrast or pre-filtering strategies.

RQ3: *What trade-offs are there between effort reduction and accuracy?* – While the post-filtering strategies attain a very high ER scores (see Table VII), they perform very poorly in terms of AMI and SER. This suggests that there indeed is a trade-off between "brute force" effort reduction and accuracy, and highlights the importance of having effort reduction measures like the SER that incorporates a notion of cluster quality (Homogeneity in our case).

VII. THREATS TO VALIDITY

Construct Validity: The ground truth (i.e., log labels) that we used to evaluate our clusters was created using regular expressions that our industrial collaborator currently uses to identify whether a log concerns a known issue. A potential threat to construct validity is that these patterns do not cover all issues encountered in the full dataset (which also motivates our research). We mitigated this threat by creating a ground truth from the subset of available data that is fully covered by the patterns. A second threat is that we can not guarantee or check the absence of matching errors in these patterns. This is, to some extent, mitigated by the fact that the patterns have been used to satisfaction by our partner. A last threat to construct validity is that we employ a hard clustering approach that assigns each log to exactly one cluster that groups all logs sharing the same characteristics. This means the approach cannot distinguish between individual issues in log files that contain multiple issues, but will create new cluster for all logs that share the same set of issues. We mitigate this threat by limiting the dataset to logs that match with only one issue.

Internal Validity: First, finding subtle differences between six treatments with only 18 observations per treatment is difficult. A larger dataset, ideally obtained from diverse sources, could help identify finer distinctions between the variants. In particular, for the Wilcoxon Signed-Rank test as implemented in `stats.wilcoxon`, it is recommended (though not mandatory) to have at least twenty observations whereas we have eighteen. Second, all algorithms and statistical procedures were implemented in Python and R with the help of widely used libraries such as NumPy and SciPy, and they were thoroughly tested. Nevertheless, we can not guarantee the absence of implementation errors that could have affected our results. Finally, a source of variability when comparing log clustering approaches is the log abstraction mechanism used. Our study does not assess the performance of the log abstraction tool provided by Cisco Norway against an automated abstraction mechanism as the one developed by Fu et al. [10].

External Validity: Our study evaluates LogCluster and its variations on log files that result from CDt activities at our industrial partner. These logs vary considerably in size and events covered, and should therefore provide a realistic picture of the behavior that can be expected in various contexts. Nevertheless, these logs likely do not cover all possible variation, so we cannot rule out that different logging practices in other systems or organizations would lead to different results.

VIII. RELATED WORK

This work replicates and extends Lin et al's LogCluster study [7] in the context of CE logs, as explained in Sections II and III. One of our other recent papers also takes LogCluster as a starting point, but differs in objectives from this paper in that it investigates whether either applying dimensionality reduction or selecting a different cluster merge criteria gives better results [8]. It also investigates the performance *stability* of each configuration to variations in two hyperparameters: the clustering algorithm's stopping threshold θ and a hyperparameter γ that decides the proportional weighing of an event's inverse document frequency weight and the contrast weight (see Section II) by making the final event weight $w_{(e)}$ the result of $w(e) = \gamma * w_{con}(e) + (1 - \gamma) * w_f(e)$ [8].

Shang et al. [25] attempt to reduce the effort needed to investigate errors in logs obtained from large Big Data

program runs. Their approach partitions a monolithic log into event sequences associated with a given subtask, abstracts each event sequence so that runtime- and substask-specific aspects are suppressed, and then group subtasks with identical abstracted event sequences together. These steps are performed for both development and production runs of the program being tested, and event sequences that do not appear in both environments are highlighted to the user. While this approach is similar to that of Lin et al. in the use of log abstraction and in leveraging the difference between development and production settings, it differs in only grouping together abstracted event sequences that are *identical*: This makes it reduce less effort than LogCluster, as documented by Lin et al [7].

An often pursued goal in log analysis is anomaly detection. Xu et al. [26] demonstrate a technique for automatically identifying anomalous events in large system logs. The anomalous events are presented to the user with decision trees that show why the event was flagged as anomalous. He et al. [27] present an experience report on using system log analysis for anomaly detection. They describe six state-of-the-art log-based anomaly detection methods but do not draw conclusions on which method performs best. We do not focus on anomaly detection in this paper, but on aiding troubleshooting by grouping together logs that fail for the same underlying issues.

Oliner et al. [28] give a general overview of challenges and advances in log analysis. They point out the difficulty of deriving actionable insights from statistical log analysis–a topic we wish to return to in future work (see Section IX-B).

Several works investigate techniques for aiding the analysis of Continuous Engineering information. SQA-mashup by Brandtner et al. [3] is a tool for visualizing and summarizing Continuous Engineering information, especially metrics such as test coverage. SQA-mashup generates views that can be adapted to the needs of different users, and SQA-profiles [4] investigates automatically deriving project-agnostic user profiles from version control and issue tracking data that can be used to tailor the data presentation. CIViT [29, 30] is a method for visualizing the overall testing activities of an organization in order to prioritize Continuous Engineering efforts better. It shows the testing coverage, degree of automation and test frequency employed in various parts of the system from individual components to full product releases. CIViT is also used in Cinders [31], an extended modeling technique that adds extra viewpoints illuminating the causal relationships between the modules in the system (i.ehow a successful build can trigger a test suite run) at various levels of abstraction. We see these techniques for modelling and visualizing CE data as complementary to our work, as the selected log representatives can be fed as input to these models.

Overall, the literature on adopting CE practices frequently alludes to the need for systematic analysis of the data generated in CE [1–6], with analysis of test results considered one of the most critical issues by practitioners [1]. Indeed, Shahin et al. [32] identify the lack of transparency and awareness regarding buid and test results as a main barrier to adopting CDy practices, together with the need for tools to help developers coordinate their debugging efforts. This need for debugging assistance is also highlighted by Hilton et al. [2]. Similar concerns came up in our in discussions with Cisco Norway, which led us to investigate methods for complementing and extending their existing regular expression-based methods with automatic clustering.

IX. Concluding Remarks

A. Contributions

Making sense of failing test run logs as effectively as possible is essential for expediently correcting software errors. To this end, we have looked at the technique of clustering failing test run logs together, so that operators can save time by inspecting a single representative log from each cluster rather than all logs. The starting point for our investigation has been a replication and extention of earlier work on system log clustering by Lin et al. [7]. Our extensions have investigated whether variations in either the event frequency weighing scheme or the method for leveraging differences between failing and passing logs could yield more accurate clusterings and a higher effort reduction. To this end, we have empirically investigated the performance of a total of six variants including the variant proposed by Lin et al. [7] on a dataset provided by our industrial collaborator Cisco Norway. Our study has focused on three research questions:

(1) Can we reduce the effort needed to discover all latent issues in a set of failing runs?
(2) How to best leverage the contrast between passing and failing runs to increase accuracy?
(3) What trade-offs are there between effort reduction and accuracy?

The results allow us to answer these questions as follows:

(A1) The LogCluster baseline achieves a *raw* median effort reduction score of $.8495$. Scaling this number by the Homogeneity of the obtained clusters, we obtain a scaled effort reduction score of $.5874$. So while the effort reduction can be considerable, the user should not exclusively rely upon this tool when troubleshooting issues, as there is a chance for false positives.
(A2) Using Adjusted Mutual Information (AMI) as quality measure, our experiments could not determine any significant difference between the proposed variants. However, when using scaled effort reduction as quality measure, our tests suggests that the *post-filtering* variant performs significantly worse than the other variants.
(A3) There are clear trade-offs between the raw effort reduction (ER) measure and accuracy (AMI and H). In our tests, this is clearly seen by how the post-filtering variants are clear winners on raw effort reduction, but clear losers on the other quality measures. This also highlights the importance of adjusting the recorded effort reduction by the quality of the resulting clusters.

Conclusion: We conclude that problem identification via automated log clustering in a continuous engineering context is viable and promises a significant effort reduction. Moreover,

the alternative clustering pipelines investigated do not yield significant advantages over the original LogCluster pipeline.

B. Future work

There are several directions in which our work can be extended. The most important is the evaluation of these techniques on a wider range of case studies. A challenge in that respect is obtaining a solid ground truth for evaluating the results, as fully labelled collections of log files are rare. One way to address this challenge is to programmatically synthesize labeled log files with known characteristics.

A second direction concerns understanding the issue that is captured by a cluster of log files. In LogCluster this is based on selecting a representative log for each cluster. However, such a representative log may still be considerably long and difficult to diagnose. We want to investigate summarization techniques that use information from the logs in a cluster to create a concise summary of the issue captured by that cluster.

Last but not least, a hard clustering approach like LogCluster cannot distinguish between multiple issues in a log file, as discussed in the threats to validity. It would be interesting to investigate how well soft/fuzzy clustering approaches, that allow a log to become a member of several clusters, can be adopted for clustering-based problem identification of CE logs.

Acknowledgements: This work is supported by the Research Council of Norway through the Certus SFI (#203461/030). We thank Marius Liaaen and Thomas Nordnes of Cisco Systems Norway for extensive discussions, help with obtaining and understanding the data set, and for developing the log abstraction mechanisms.

References

[1] E. Laukkanen, J. Itkonen, and C. Lassenius. "Problems, causes and solutions when adopting continuous delivery - A systematic literature review." In: *Information and Software Technology* 82 (Feb. 2017), pp. 55–79.

[2] M. Hilton, N. Nelson, T. Tunnell, D. Marinov, and D. Dig. "Trade-offs in continuous integration: assurance, security, and flexibility." In: *Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)*. New York, New York, USA: ACM, 2017, pp. 197–207.

[3] M. Brandtner, E. Giger, and H. Gall. "SQA-Mashup: A mashup framework for continuous integration." In: *Information and Software Technology* 65 (Sept. 2015), pp. 97–113.

[4] M. Brandtner, S. C. Muller, P. Leitner, and H. C. Gall. "SQA-Profiles: Rule-based activity profiles for Continuous Integration environments." In: *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, Mar. 2015, pp. 301–310.

[5] A. Debbiche, M. Dienér, and R. Berntsson Svensson. "Challenges When Adopting Continuous Integration: A Case Study." In: *Lecture Notes in Computer Science (LNCS)*. Vol. 8892. Springer, 2014, pp. 17–32.

[6] H. H. Olsson, H. Alahyari, and J. Bosch. "Climbing the Stairway to Heaven - A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software." In: *Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, Sept. 2012, pp. 392–399.

[7] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen. "Log Clustering Based Problem Identification for Online Service Systems." In: *International Conference on Software Engineering - Software Engineering in Practice (ICSE SEIP)*. New York, NY, USA: ACM, 2016, pp. 102–111.

[8] C. M. Rosenberg and L. Moonen. "Improving Problem Identification via Automated Log Clustering using Dimensionality Reduction." In: *International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, 2018, Article No. 16.

[9] C. D. Manning, P. Raghavan, H. Schütze, et al. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[10] Q. Fu, J.-G. Lou, Y. Wang, and J. Li. "Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis." In: *International Conference on Data Mining*. IEEE, Dec. 2009, pp. 149–158.

[11] N. X. Vinh, J. Epps, and J. Bailey. "Information theoretic measures for clusterings comparison." In: *International Conference on Machine Learning (ICML)*. ACM, 2009, pp. 1073–1080.

[12] A. Rosenberg and J. Hirschberg. "V-measure: A conditional entropy-based external cluster evaluation measure." In: *Joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*. Association for Computational Linguistics, 2007, pp. 410–420.

[13] M. Friedman. "The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance." In: *Journal of the American Statistical Association* 32.200 (1937), pp. 675–701.

[14] J. Demšar. "Statistical Comparisons of Classifiers over Multiple Data Sets." In: *Journal of Machine Learning Research* 7 (Aug. 2006), pp. 1–30.

[15] F. Wilcoxon. "Individual Comparisons by Ranking Methods." In: *Biometrics Bulletin* 1.6 (1945), pp. 80–83.

[16] J. W. Pratt. "Remarks on Zeros and Ties in the Wilcoxon Signed Rank Procedures." In: *Journal of the American Statistical Association* 54.287 (Sept. 1959), p. 655.

[17] A. Benavoli, G. Corani, and F. Mangili. "Should we really use post-hoc tests based on mean-ranks?" In: *Journal of Machine Learning Research* 17.5 (May 2016), pp. 1–10.

[18] S. Holm. "A Simple Sequentially Rejective Multiple Test Procedure." In: *Scandinavian Journal of Statistics* 6.2 (1979), pp. 65–70.

[19] H.-P. Piepho. "An Algorithm for a Letter-Based Representation of All-Pairwise Comparisons." In: *Journal of Computational and Graphical Statistics* 13.2 (June 2004), pp. 456–466.

[20] A. Vargha and H. D. Delaney. "A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong." In: *Journal of Educational and Behavioral Statistics* 25.2 (June 2000), pp. 101–132.

[21] E. Jones, T. Oliphant, P. Peterson, et al. *SciPy: Open source scientific tools for Python*. 2017.

[22] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009.

[23] R Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria, 2016.

[24] S. Graves, H.-P. Piepho, and L. S. with help from Sundar Dorai-Raj. *multcompView: Visualizations of Paired Comparisons*. 2015.

[25] W. Shang, Z. M. Jiang, H. Hemmati, B. Adams, A. E. Hassan, and P. Martin. "Assisting developers of Big Data Analytics Applications when deploying on Hadoop clouds." In: *International Conference on Software Engineering (ICSE)*. IEEE, May 2013.

[26] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan. "Detecting large-scale system problems by mining console logs." In: *Symposium on Operating Systems Principles (SOSP)*. ACM, 2009, p. 117.

[27] S. He, J. Zhu, P. He, and M. R. Lyu. "Experience Report: System Log Analysis for Anomaly Detection." In: *IEEE International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, Oct. 2016, pp. 207–218.

[28] A. Oliner, A. Ganapathi, and W. Xu. "Advances and challenges in log analysis." In: *Communications of the ACM* 55.2 (Feb. 2012), pp. 55–61.

[29] A. Nilsson, J. Bosch, and C. Berger. "Visualizing Testing Activities to Support Continuous Integration: A Multiple Case Study." In: *Agile Processes in Software Engineering and Extreme Programming*. Springer, 2014, pp. 171–186.

[30] A. Nilsson, J. Bosch, and C. Berger. "The CIViT Model in a Nutshell: Visualizing Testing Activities to Support Continuous Integration." In: *Continuous Software Engineering*. Springer, 2014, pp. 97–106.

[31] D. Ståhl and J. Bosch. "Cinders: The continuous integration and delivery architecture framework." In: *Information and Software Technology* 83 (Mar. 2017), pp. 76–93.

[32] M. Shahin, M. Ali Babar, and L. Zhu. "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices." In: *IEEE Access* 5 (2017), pp. 3909–3943.