Panning Requirement Nuggets in Stream of Software Maintenance Tickets

Senthil Mani, Karthik Sankaranarayanan, Vibha Singhal Sinha
IBM Research, India
{sentmani, kartsank, vibha.sinha}@in.ibm.com

and Premkumar Devanbu
University of California, Davis
ptdevanbu@ucdavis.edu

ABSTRACT

There is an increasing trend to outsource maintenance of large applications and application portfolios of a business to third parties, specialising in application maintenance, who are incented to deliver the best possible maintenance at the lowest cost. To do so, they need to identify repeat problem areas, which cause more maintenance grief, and seek a unified remedy to avoid the costs spent on fixing these individually. These repeat areas, in a sense, represent major, evolving areas of need, or requirements, for the customer. The information about the repeating problem is typically embedded in the unstructured text of multiple tickets, waiting to be found and addressed. Currently, repeat problems are found by manual analysis; effective solutions depend on the collective experience of the team solving them. In this paper, we propose an approach to automatically analyze problem tickets to discover groups of problems being reported in them and provide meaningful, descriptive labels to help interpret these groups. Our approach incorporates a cleansing phase to handle the high level of noise observed in problem tickets and a method to incorporate multiple text clustering techniques and merge their results in a meaningful manner. We provide detailed experiments to quantitatively and qualitatively evaluate our approach.

Categories and Subject Descriptors

K.6.3 [**Software Management**]: Software Maintenance

Keywords

Text Clustering, Requirements, Mining Software Repositories

1. INTRODUCTION

Software Maintenance is a crucial phase of any software project that continues long after the product has been developed. There have been a number of estimates of effort spent in software maintenance and enhancement. Figures varying from 40-80 percent of total systems and programming resources have been cited [13]. In some market segments, the maintenance phase of the software lifecycle has undergone dramatic changes. Rather than undertake this costly task in-house, many large and small businesses have taken to outsourcing their maintenance activities. Market research firms estimate the *Outsourced Software Maintenance* business to be around $250 Billion a year business as of 2012 [18].The software is turned over (complete with source code, manuals, data, etc) to a *maintenance service provider* (such as IBM, Accenture, Cognizant, Infosys, TCS, *inter alia*).

The practice of outsourced software maintenance differs substantially from the traditional in-house version. The outsourced service provider typically *operates* the system, in addition to providing maintenance. Maintenance in this setting consists of answering support calls, and issuing and handling trouble tickets. This is a challenging and dynamic business. Service providers compete on the basis of low costs, and negotiated service-level agreements (SLAs), which impose volumes and deadlines on the resolution of trouble tickets. In a typical setting, a service provider handles hundreds of calls a day; hundreds or thousands of trouble tickets are issued and resolved on a weekly basis. Cost, and the honouring of SLAs are of paramount importance. Tickets are typically handled one by one; however, groups of tickets often follow common patterns (examples below). Recognising these patterns and exploiting common synergies with a unified response can lead to substantial cost, quality and interval benefits. The response could include: process changes that alter the way certain tickets are handled, changes to documentation, automating a frequently used service, and also changes to code to add new features. The recognition of common patterns is both important and very challenging. In practice, managers are dependent on experience of individuals resolving these tickets to recognize and report these patterns on an ad-hoc basis. In a sense, the emerging patterns in the tickets indicate a kind of "requirements drift" that is latent in the otherwise undifferentiated mass of individually-generated tickets; these patterns must be extracted somehow to allow efficient and co-ordinated responses.

Consider the example of a specific E-Commerce web-site[1], maintained by IBM. It allows users to log in using a username and password. The site did not provide a feature to reset the password and hence, anytime a user forgot his/her password they would open a problem ticket. Someone in the maintenance team would then reset the password. In the first year of the web-site there were only 5 tickets related to "password reset". However, as the user base increased the number of password resets increased to 100. This was noticed informally by alert personnel. The cost of enhancing the application to provide a self password reset option was 10,000$. However, the cost of manual password reset was only 10$. Hence, the team

FSE'14, November 16–21, 2014, Hong Kong, China
Copyright 2014 ACM 978-1-4503-3056-5/14/11...$15.00
http://dx.doi.org/10.1145/2635868.2635897

[1]Identifying information omitted for confidentiality.

decided to not implement the feature. In the subsequent year, the password resets increased to 1500 and now the work-around was not cost-effective anymore and hence the feature was supported.

Consider another example from a supply chain application portfolio maintained by IBM. A client owned several warehouses and trucks delivering goods between them. Each truck was fitted with a GPS device which on returning to its base warehouse, triggered an alert that updated a database record marking the truck as available. During the day to day operations, the application maintenance team were receiving about 20 tickets per month, each asking for a manual release of trucks. Consultants were able to respond to this quickly (in less than 10 mins) by manually running a simple query to update the database record, and thus close these tickets. However, considering that an automatic GPS-based solution was already in place, these tickets should not have arisen in the first place. The repetitive pattern of problem tickets asking for a "vehicle release" pointed towards a bug in the application that should have been spotted, investigated and resolved.

In first example, to reduce maintenance cost, the management team needs to be able to identify "password reset" as a common pattern because as we saw, such commonly occurring problem patterns are likely to translate into requirements later on. In the second example, the management team needs to be able to identify "vehicle release" as a common pattern, as this problem indicated a systematic application fault.

Our research goal is to build tools that assist managers in identifying these emerging common patterns from the corpus of past tickets. Towards this end, we make the following contributions to help outsourced maintenance software projects manage large volumes of trouble tickets:

- Tickets contain information about the problem in its title and description as natural language text. We develop a text clustering method that identifies emerging patterns of tickets that can be addressed together in a unified manner.[2]
- It is imperative that maintenance teams are able to quickly validate whether the identified patterns indicate valid requirements for the maintained system. Hence, the labels used for summarizing the patterns become very important. We build a phrase-based approach to assign labels to the identified ticket clusters in a way that would be strongly suggestive of unified responses, and thus be salient to practitioners.
- We evaluate the quality of the ticket clusters we find, using an entropy-based measure of assigning tickets to people (using historical data) and using qualitative methods on 6 large application maintenance projects within IBM.

This need to mine common problem patterns to identify feature requests has also been seen in open source projects. In [5, 11], authors have presented approaches based on text mining to identify problem patterns from support forums. In [11], authors apply their proposed technique on Firefox forum entries and found problem clusters indicative of technical challenges users are facing that could be addressed with enhancements. They report that Firefox Principal Engineer felt that this kind of approach would help their open source community to prioritise their efforts around commonly reported challenges. In [5], the authors have applied an existing clustering algorithm to group forum entries requesting for same/similar feature. They present a prototype of their approach and evaluate it on support forum for an open source project called SugarCRM. They also report that this approach helps the technical team easily identify new feature requests being asked for by user community. Similar to the need in outsourced maintenance projects, both these works underscore the need of a technique that can help technical team easily identify requirements for feature requests and enhancement from textual data. However, both of these papers present very limited evaluation. We build upon approaches suggested in these papers to build a system that uses a combination of different text clustering techniques to identify cohesive problem clusters. As far as we know, our work is the first one that presents detailed quantitative and qualitative experiments that help evaluate the efficacy of the approach on large outsourced maintenance projects.

[2]Note that in this paper, we are *not* proposing a new general-purpose text clustering algorithm, but rather, a method to cluster trouble tickets using multiple existing ideas from text mining so as to effectively identify emergent patterns of requirements from them.

Software engineering community has experimented with use of text analysis for multiple other use-cases. For example, duplicate bug report detection [21, 15], bug assignment [2], finding similar bugs [3] to help in fault localization. The underlying approach here is to identify past bug reports that are textually similar to the given bug report. However, because the end use-case is different, these approaches focus on different design considerations than our approach. For e.g. in the case of duplicate bug report detection, it is imperative, that there exist a very high textual similarity between old and the new bug report. Hence, precision of analysis is of paramount importance. Also, finding one past similar bug report is sufficient rather than a collection of similar reports. In case of bug assignment, rather than text clustering, classification based approach works well, where each developer becomes a class that can be learnt based on historical data. A new bug report is auto-assigned to one of these classes. Classification approaches do not work in our use-case, as the class of problem categories is not known upfront and need to be discovered from ticket data.

Rest of the paper is organized as follows. In next section we present important design points for the approach based on the use-case need and nature of enterprise problem ticket data. In Section 3, we present our approach, followed by 7 experiments that measure different aspects of the approach in Section 4. Multiple teams within IBM are using our approach. In 4, we present examples from two IBM teams that have successfully used our system to identify actionable requirements. Finally, we present related work in Section 5 and conclusions in Section 6.

2. DESIGN CONSIDERATIONS

When compared to typical product maintenance, problem ticket repositories in outsourced maintenance projects contain data corresponding to both, so called *bugs* (managed in open source projects using bug tracking systems such as *Bugzilla*, *Jira* etc.), and *support requests* (managed using discussion forums in open source). The nature of unstructured text in problem tickets could be 1) automatic machine-generated text - this type of text is generally well formatted arising from alerts put in applications to indicate issues, 2) human-generated free text directly typed into these unstructured fields - their informativeness could vary from being terse to being very detailed, 3) copy-pasted text from emails threads and other sources of discussion, or any combination of the above types. (See Table 1 for examples.) We now identify a couple of major issues that must be addressed when grouping these problem tickets into unified, meaningful groups (or clusters).

1) *Readable Cluster Labels*: Application maintenance practitioners need a quick interpretable way to understand what each cluster represents as they do not have enough time to peruse each individual ticket in a group. In Table 2, we show some examples of readable labels that are easier to comprehend versus labels that are not self explanatory. Later in the paper, in experiment 7, we present using a user-study the importance of this design consideration in overall usefulness of the approach.

Table 1: Examples show variance in nature of ticket text.

Machine-generated : well-formatted
US5163WEBP APDReports 239499: System.NullReferenceException: Object reference not set to an instance of an object.
US0894SRAP R3Abap CRITICAL: 'R3Abap Shortdumps Frequency-:Frequency 14 /min > 10 /min (no. of messages per minute above threshold)
US0726BIZP BTSAPShr 48743: Document(s) have failed. Please use the BizAdmin Application for more information. (ErrMonitor)
Natural Language : conversational and noisy
To: support@company.com Subject: Please help John Doe and I upgraded our NVS Portal to version 5 and we are unable to open the <company> Daily Volumes app. We get the message showing a Backend connection error which we have never seen before in this context. ... I removed the QV plugin from memory and tried again but received the same message connection error, and this time with an empty response. I'm not sure how to proceed. Thanks, Jane Doe

2) *Variance in unstructured nature of text* : The overall clustering approach should be able to handle the variance in unstructured nature of the text (from machine generated to human entered free text) and maximize the number of tickets clustered, without necessarily having to know beforehand the nature of data in each ticket. Later in the paper, in experiments 5 and 6, we evaluate the individual steps used in our approach that were developed to handle the nature of text and how they significantly improved the problem cluster quality.

Having characterized the nature of the unstructured data and the corresponding requirements, we next provide a detailed description of the approach we developed for our task.

3. PROBLEM TICKET CLUSTERING (PTC)

We call our approach *Problem Ticket Clustering* (PTC). The first phase of this approach is Data Cleansing (Sect. 3.1) to prepare the data for the remaining phases. This phase takes the raw unstructured data and removes unnecessary information such as email headers, signature blocks, greetings, etc. Such noise severely impacts the performance of the following phases and hence to obtain meaningful results, we define certain pre-processing rules for data cleansing specific to the nature of problem tickets.

The second phase of our approach is the Grouping phase (Sect. 3.2) which takes as input the cleansed data and produces as output clusters of problem tickets based on similarity of problems faced along with meaningful labels describing these clusters. We employ two distinct clustering algorithms with complementary capabilities, i) a latent semantic indexing based technique called Lingo [17, 22] which works well with large non-regular unstructured text and is known to provide good-quality meaningful cluster labels, and ii) a novel hierarchical n-gram based technique which is designed to work well with short snippets of well-formatted text. We then develop a cluster merging technique to merge the results from these two complementary algorithms. We next describe each of these phases in detail.

3.1 Data Cleansing and Noise Removal

Typically, raw data in unstructured fields contain conversations (email threads, etc.) and therefore can have a lot of noise which hamper text clustering. To handle this noise, we employ the following heuristics and cleanse the data before presenting it for the grouping phase.

Heurisitic 1: Removal of Email-specific noise. Since the data can contain copy-pasted email threads, we employ regular expressions to remove noise arising from them. We remove email headers (FROM:, TO:, CC:, BCC:, SUBJECT:, etc.), commonly used salutations (HI, HELLO, DEAR, HEY), signature blocks that usually occur at the end of email messages, and email formats such as *abc@foo.com* - regular type, *<Name>/<Country>/<Company>* - enterprise type. This is similar to heuristic 1 proposed in [19].

Heuristic 2: Removal of Context-specific Stop Words. We use a list of words provided by user (based on domain knowledge) as stop words over and above the generic English stop words list that many other techniques also employ. For example, in a problem ticket collection dealing with SAP issues, the word "SAP" occurs very frequently but is uninformative.

Heuristic 3: Removal of Names. People names in text typically do not help discern the technical problems; but their frequent occurrence can hinder clustering. Hence we remove them by providing the ability to specify the names if available in the ticket dump in columns such as RESOLUTION OWNER, CUSTOMER, etc. as these are the names commonly observed in the ticket descriptions.

Heuristic 4: Misc. Regular Expression-based Heuristics. On top of the above mentioned heuristics, we also employ regular expressions to remove entities such as URLs, dates and numbers.

3.2 Grouping Tickets into Labeled Clusters

The grouping phase is divided into the following steps. In the first step, we extract frequently occurring phrases in the ticket text and then filter them to derive a pool of candidate cluster labels (Sect. 3.2.1). Next, to group the tickets, we employ multiple clustering techniques to handle the variance in the unstructured nature of the data and build a robust clustering method (Sect. 3.2.2). In addition to a standard, off-the-shelf vector-space clustering technique, we introduce a novel *hierarchical n-gram technique* to deal with the presence of machine-generated trouble tickets along with human-produced ones. Finally, we apply a merging algorithm to combine the results of these multiple techniques to present meaningful, well-labeled groupings of tickets which are agnostic to the nature of ticket data present in the data set (Sect. 3.2.3). In the following sections, we describe each of the above steps.

3.2.1 Candidate Label Generation

The first step is to extract frequently appearing phrases in the document set which can later be used to induce labels for the clusters formed. A phrase is a sequence of words occuring together, commonly refered to as n-gram, where *n* indicates length of phrase. To obtain readable labels, we seek to detect complete phrases in the input text which: i) are shorter than a specified maximum term length, ii) occur at least a specified number of times in the document set, iii) do not cross sentence boundaries (such as periods), and iv) discount the presence of certain words in the subsequence (words that occur too frequently). These filtered set of phrases constitute the *candidate set* of labels for the following clustering step. To illustrate this technique, consider the following example sentence.

Example Sentence: *Inbound GLMAST Idoc failed*
Bigrams (2 word phrases): *Inbound GLMAST, GLMAST Idoc, Idoc Failed, Inbound Idoc*
Trigrams (3 word phrases): *Inbound GLMAST Idoc, GLMAST Idoc failed, Inbound Idoc Failed*
4-gram (4 word phrases): *Inbound GLMAST Idoc failed*

Table 2: Comparison of labels from PTC and LDA.

Labels from PTC	Labels from LDA
Change Master Workflow Errors	master amd workflow wifi
Removal of Duplicate Accounts	ms unable account error
Role Authorization Issues	run auth max role
Daily Monitoring Errors	netscape tools error monitor
Batch Job Failures	job service failed software

Note that here, the word *GLMAST* is known as a context-specific stop word and hence its presence is considered optional when calculating candidate labels. Assuming that the word *Idoc* occurs too frequently in the tickets collection, the only label remaining in the candidate set would be *Inbound failed*. We use a minimum phrase length of 2 and maximum of 10.

One can argue that we can group the problem tickets simply by using these candidate labels, and present the groups to the end-users. However, there are two main problems: i) frequent phrases based grouping will result in large number of groups which will be laborious and frustrating for the practitioners to act upon, and ii) there will be a large overlap of problem tickets across groups, and therefore the very purpose of grouping them is then lost. Text Clustering seeks precisely to address these problems by considering the text as whole and leverages the notions of similarity (and distance), and groups tickets into clusters with minimal overlap.

3.2.2 Text Clustering

Text Clustering is essentially an unsupervised learning approach. It is well understood in the text mining community that no single learning technique can work well for all types of data, but rather individual methods have to be picked and tuned to meet specific requirements [23]. The K-means clustering algorithm [9] which was popular for its simplicity a decade ago has been superseded by methods which capture the semantic relations between words in a document collection by performing Latent Semantic Analysis [12]. Approaches based on Topic modelling that are commonly used these days such as Latent Dirichlet Allocation (LDA) [4] tackle the labelling requirement post-facto by generating high probability terms associated with each cluster and use them to assign labels to clusters. But such labels are hard-to-interpret and lead to poorly-labeled clusters (see Table 2).

Keeping in mind the limitations of these techniques and considering our requirements, we employ the following two complementary clustering techniques, i) Lingo [17]: a popular technique that combines the results of frequent phrase finding with the strength of latent semantic analysis to identify best labels representative of the document set involving long non-regular text in natural language (well suited for human authored tickets), and ii) a novel hierarchical technique based on identifying frequently occurring n-grams (sequences of tokens of length n) in the data set which is particularly suited for short, well-formatted text (such as machine generated error logs). Humans could have used different words and sentence formations to describe similar problems. So a clustering technique that converts the problem text into bag of words (such as *Lingo*) is likely to work well. When the problem ticket is generated by a machine, the sentences are very regular, so a language model based approach that focuses on sequence of words (such as *n-grams*) is likely to give better results.

As a common pre-processing step for both techniques, we run the entire ticket text through a stemming and stop word removal operation [11], which is a standard linguistic technique applied in most text processing tasks. We next describe the two methods and use the two example ticket sets (Table 3) to illustrate the complementary strengths of each of the techniques.

Table 3: Examples to highlight the benefits of complementary clustering techniques

Example Set 1:
T11: *My annual leave verification.*
T12: *Unable to apply leave. Leave forecast balance is Zero*
T13: *I can't apply for leave and why I cant find annual leave balance in Pso*
T14: *annual leave application error*
T15: *Discrepancy in my leave balance.*
For the above tickets, we would expect to get the following 3 clusters:
Cluster 1: T11, T13, T14 regarding annual leave
Cluster 2: T12, T13, T15 regarding leave balance
Cluster 3: T12, T13 regarding apply leave

Example Set 2:
T21: *GMP: Failed ABC:aaa document*
T22: *IMPL: Failed DEF:ddd document*
T23: *BLAST: Failed XYZ:xxx document*
T24: *GMPL: Failed XYZ document*
We would expect to get the following clusters:
Cluster 1: T21, T22, T23, T24 regarding Failed Document
Cluster 2: T23, T24 regarding Failed XYZ Document

Lingo Clustering: In what follows, we use the term *document* to describe the unstructured text data from a single problem ticket and *document set* to describe the entire ticket dump. First, the document set is represented as a term-document matrix A, where the columns of the matrix correspond to terms appearing in the documents, and rows of the matrix correspond to the documents. The value in each cell is proportional to the frequency of a term in that particular document and inversely proportional to its frequency in the entire document set (tf-idf). This matrix A is now decomposed using Singular Value Decomposition (or any other method such as Non-negative Matrix Factorization) to obtain an orthogonal basis of vectors in the feature space as specified in Eqn. 1

$$A = U * S * V^T \tag{1}$$

where S is the diagonal matrix of singular values, and U and V are the left and right matrices of orthogonal bases vectors.

Next, the *candidate set* of labels discovered previously (Sect. 3.2.1) are expressed in the same term-document space by considering them as tiny documents themselves, with each candidate label represented as a document constituted by the terms in the label. After this, the similarity of each such candidate label is calculated with respect to the first k orthogonal basis vectors as shown in Eqn. 2

$$M = U_k^T * P \tag{2}$$

where P is matrix of documents formed by the candidate set of labels and M is the matrix of similarity values between each candidate label in P and each of the top k basis vectors in U calculated using the cosine similarity function as

$$\mathrm{sim}(u,p) = \frac{\mathbf{u} \cdot \mathbf{p}}{\|\mathbf{u}\| \, \|\mathbf{p}\|} = \frac{\sum_{i=1}^{n} w_{i,u} w_{i,p}}{\sqrt{\sum_{i=1}^{n} w_{i,u}^2} \sqrt{\sum_{i=1}^{n} w_{i,p}^2}} \tag{3}$$

where $w_{i,u}$ and $w_{i,p}$ represent the weights of base vector u and candidate label vector p respectively.

Now, using these cosine similarity values, a label is assigned to each of the top k basis vectors based on the one that is most similar to them from the *candidate set*. The final step is to assign documents to these basis vectors by calculating their cosine similarity with these vectors, and assigning them to those vectors and labels which fall within a threshold. Note that, this could result in some tickets being assigned to multiple vectors, but this is preferred over picking only the closest one (which would be sensitive to relative similarities between documents).

Therefore, this entire process results in grouping the documents according to orthogonal basis vectors that guarantee diversity of topics covering the term-document space and are also attached with most frequently occurring phrases that are most similar to these document clusters, thus resulting in well-labeled clusters.

Applying this clustering technique on *Example Set 1* yields the following labeled clusters:

Cluster 1: Label - *Annual Leave*, Tickets - T11, T13, T14.
Cluster 2: Label - *Leave Balance*, Tickets - T12, T13, T15
Cluster 3: Label - *Apply Leave*, Tickets - T12, T13

which is in line with what we would like to obtain. However, on *Example Set 2* it yields only the following cluster:

Cluster 2: T23, T24 with Label Failed XYZ Document

It fails to identify Cluster 1 since "Failed, document" occurs in all the tickets. Therefore each document's column entries corresponding these terms receive low weights due to the idf factor (in tf-idf weighting explained above), and this results in this cluster not being identified. We therefore introduce a new clustering technique based on n-grams, *Hierarchical n-gram clustering*

Hierarchical N-gram Clustering Technique: As mentioned before, to handle machine-generated text which are well-formatted (arising from templates and therefore exhibit almost exactly repeating phrases), we require a technique that focuses on extracting frequently appearing n-grams. For this we develop a suitable hierarchical n-gram clustering technique as follows.

The first step in this technique is to filter the set of phrases that will be used to create the clusters. To do so, we first identify all bigrams extracted in the candidate label generation phase and then for each bigram, form a cluster from all the tickets where this bigram appears. This could result in a cluster of tickets being associated with multiple bigrams (due to multiple common phrases). Therefore, to decide which of the associated bigrams to choose as the label for such a cluster, we pick the bigram having a higher *log likelihood ratio*. The log-likelihood ratio measures the deviation between the observed data "*word1 word2*" and what would be expected if *word1* and *word2* were statistically independent. The higher the score, the less evidence there is in favour of concluding that the words are independent. It is calculated as follows:

$$LL = 2(n_{11}\log\frac{n_{11}}{m_{11}} + n_{12}\log\frac{n_{12}}{m_{12}} + n_{21}\log\frac{n_{21}}{m_{21}} + n_{22}\log\frac{n_{22}}{m_{22}}) \tag{4}$$

where n_{11} is the number of times *word1 word2* occur together, and n_{12} is the number of times *word1* occurs with some word other than *word2*, and so on. Here the m_{ij} values are the *Expected Values* of the words occurring together and are calculated by taking the product of their associated marginals and dividing by the sample size, as follows:

$$m_{11} = \frac{n_{1p}n_{p1}}{n_{pp}} \tag{5}$$

where n_{1p} is the number of times in total that *word1* occurs as the first word in a bigram, and so on. The formula is similarly extended for tri-grams, 4-grams, 5-grams and so on.

With this, each cluster now has exactly one label. We then put each ticket in the clusters whose labels it contains. Similarly, we repeat the above process using the tri-grams as the candidate phrase set, 4-grams and so on. At the end of this process, we get a hierarchy of clusters where each ticket would belong to a bigram cluster and could further also belong to child trigram, 4-gram and so on clusters.

To illustrate the working of this technique, consider the tickets in *Example Set 2*. Here, the two bigrams in the candidate set are *failed document* and *failed xyz*, but the only cluster generated from bigrams is *failed document* because all tickets in *failed xyz* are subsumed in it. The tri-gram step generates a cluster containing tickets *T23, T24* with the label *failed xyz document*.

Cluster 1: T21, T22, T23, T24 with Label Failed Document
Cluster 1.1: T23, T24 with Label Failed XYZ Document

thus conforming to what we would like to obtain. Applying the same technique on *Example Set 1* yields the following labeled clusters:

Cluster 1: Label - *Annual Leave*, Tickets - T11, T13, T14.
Cluster 2: Label - *Leave Balance*, Tickets - T13, T15
Cluster 3: Label - *Apply*, Tickets - T12, T13

It can be observed that T12 which is also about *leave balance* is missing from Cluster 2 because the actual phrase occurring here is *leave forecast balance*. This shows that this technique works well when the text to be clustered is concise and template based. Hence, in our overall approach we only employ it when the text length is restricted to a few sentences.

Looking at the results with the two examples sets, we observe the complementary advantages of the two clustering techniques, which together provide us with the necessary robustness to deal with the variance in the unstructured nature of the ticket text.

Both our clustering techniques require a minimum cluster size to be specified as an input. In practise, we observe that projects choose to take an action only when a problem has repeated a minimum number of times, and therefore we allow project teams to specify this value based on their specific needs. For purposes of experiments in this paper, we fix this value to be 25. Also, we employed the open-source implementation of Lingo from Carrot2 (project.carrot2.org) and retained the default value of K (200). No calibration was needed.

3.2.3 Cluster Merging

In this phase, we combine the outputs of the two clustering techniques to present a unified clustering result by merging the clusters and their corresponding labels using the following heuristics:

Minimum Cluster Merge Threshold: If the percentage of overlap in tickets across clusters from the two clustering techniques is high enough (employing a threshold of 80%), we merge the clusters into a single cluster by forming a union set of tickets from the two clusters. To pick the label of the new merged cluster, we simply pick the cluster label with higher word count.

High Similarity between Labels: It is quite commonly observed in our ticket data that same typographical errors repeated by the same practitioners lead to distinct clusters varying only by the difference of the typo. Another reason could also be due to variations in spellings employed. For example, a cluster labeled as "Job Canceled", and another labeled as "Job Cancelled" essentially represent the same underlying problem. In such cases, we find the Levenshtein distance [14] between the labels normalized by the length of the longer of the two labels, and check if it lies within a threshold (we pick 0.2). If so, we collapse the two clusters into a single cluster and pick one of the two labels as the label for the merged cluster. Note that, to avoid merging distinct labels differing by a small string

Table 4: Details of the subjects used in our experiments

Sub--jects	Total Tickets	Total Words	Avg Words	Period (mths)	Total Assignees	Max Entropy
(1)	12387	56963	4.6	6	28	5
(2)	6364	47932	7.5	20	122	7
(3)	12552	220055	17.5	6	64	6
(4)	9224	90067	9.7	14	124	7
(5)	22761	221006	10.0	27	270	9
(6)	17499	278370	16.0	24	280	9

difference not due to an error (e.g. "Cost Error" vs "Post Error"), we filter the words through an English language dictionary list. Additionally, clusters with labels where the order of words is flipped (e.g. "Job Failed" & "Failed Job") are collapsed into one. Therefore, at the end of this cluster merging technique, we obtain a unified set of document clusters along with their corresponding labels.

We now proceed to provide detailed experimental evaluation of the entire approach in the next section.

4. EXPERIMENTS

This section is organized as follows. We first describe the different datasets used for the experiments. We then describe in detail the experimental methodology employed here to evaluate and compare the proposed approach with existing techniques, and finally discuss the results obtained.

4.1 Datasets

For our evaluation we used two different datasets. The first is a small, hand-labeled dataset used to perform various experiments and compare performance against ground truth. The second dataset is a large collection of problem tickets from 6 different customer projects currently being maintained by IBM.

DS1: Hand-labeled Data: We built a dataset where tickets were collected from 11 different problem types (classes) manually marked by a Subject Matter Expert (SME) across multiple IBM clients. In order to avoid any biases due to imbalanced clustering, we collected approximately the same number of tickets from each problem type ($\sim$100 tickets each), giving us a total dataset size of 1084 tickets. This task was performed by one SME in approximately 4 hours.

The objective was to measure if the composition of clusters inferred from our approach matched the manually marked classes. This dataset was used for experiments 1, 5 and 6. To measure the performance for each of the experiments, we compared the clustering results with the manually marked classes and calculated the precision and recall of the groupings as follows. For each discovered cluster, we assign the class to which maximum tickets in it belong as its primary class. For precision, we calculated the percentage of tickets in that cluster that belong to its primary class, and average across this for all 11 classes. For recall, we calculated the percentage of all tickets from the cluster's primary class that had been detected in that cluster, and similarly average across all classes.

DS2: IBM Customer Data: We collected 80787 problem tickets from 6 live projects within IBM. To protect their identities, they have been referred to as *Subjects (1) - (6)*. Every ticket had a *Title* field containing a short description of the problem. All subjects had tickets created both automatically (by systems) and by humans. However, the proportion of auto-generated versus human created tickets varied across these subjects. In Table 4, we show the details of these subjects. We used this dataset to conduct experiments 2 and 3. The user studies for experiments 4 and 7 were also conducted by contacting practitioners working on these projects.

4.2 Methodology and Results

Through the experiments performed, we seek to answer the following three key research questions. The first question (*RQ1*) seeks to address whether our approach produces good clusters (as measured by various metrics described below) and analyzes its performance compared to existing popular clustering methods. The second question (*RQ2*) focuses on the efficacy of the specific individual steps of our approach (i.e. noise removal, cluster merging, good label generation). Finally, the third question (*RQ3*) addresses whether the application of our approach led to identification of coherent patterns that project teams acted upon to reduce their software maintenance cost.

4.2.1 *RQ1: Quality of Clustering with PTC*

To address this research question, we performed 4 experiments. In experiment 1, we used the hand-labeled dataset where tickets are associated to different classes based on the type of problem described. We ran our approach on this dataset and then calculated the *precision* and *recall* scores by comparing the clustering assignments obtained against the manually marked ground truth. This is also the evaluation approach followed in existing literature [19, 7] and possibly the most intuitive one. However, this experiment, while necessary, would not be sufficient to confirm the goodness of the clustering because the size of hand-labeled datasets are arguably small. Winbladh et al [7] also report on the challenge of creating such a truth set. In their work, it took 13.6 hours to manually label the dataset even for a small set of 327 comments. This made us further explore additional metrics to evaluate the cluster goodness.

Intuitively, tickets that indicate the same problem should be described using same or similar words and hence should show a high textual similarity when grouped together. Based on this, we developed our first new metric which we call *textual coherence*. We calculate each cluster's *textual coherence* by first calculating the textual similarity between each pair of tickets in that cluster and then compute the average across all such pairs of tickets. The text similarity metric employed here is the well-known cosine similarity [20] and is applied on the cleansed tickets. This is described in detail in experiment 2.

Next, we introduce another metric called *Assignee Entropy*. In every problem ticket, there exists a *Ticket Assignee* field which identifies the individual to whom that ticket was assigned for resolution. Typically these tickets are not assigned randomly to individuals. Rather, there is a top-level process (called "triaging") where an experienced person observes the nature of the problem and assigns the ticket to an individual based on their expertise. For example, "Password resets" are addressed by a select few amongst the entire pool of individuals. Hence, a small subset of people become specialised or trained in resolving a particular type of issue. As a result of this practice, during steady-state maintenance, each type of issue is generally assigned to those limited set of individuals who would have developed proficiency in solving that issue type. We therefore exploit this knowledge from software maintenance ecosystem that *coherent problems (problems of the same type) are solved by a limited set of individuals from amongst the entire pool*. An information-theoretic view of this would be that more coherent clusters have lower *uncertainty* about the assignment of tickets in the cluster to personnel; thus the *information content*, or *entropy* of coherent clusters should be lower.

Based on this view, we conjecture that good clusters i.e. clusters with high coherence exhibit low assignee entropy values whereas incoherent or dissimilar clusters exhibit high entropy. To mathematically capture this knowledge, if we look at a histogram of counts for the entire pool of assignees for tickets from the *same* problem type

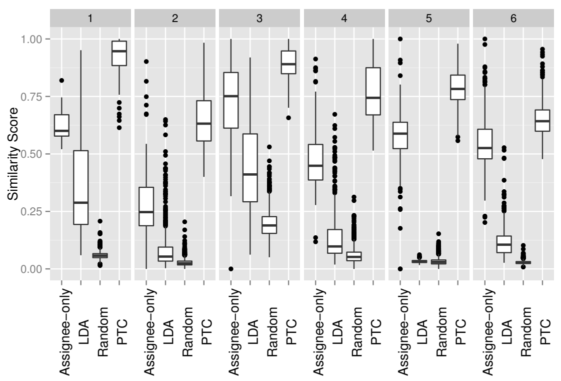

Figure 1: Textual Coherence score of clusters for different techniques - assignee-only, LDA, Random and our PTC approach

(a cluster), we should observe a distribution that exhibits low entropy (i.e. there is a fairly high certainty regarding which individuals these tickets are being assigned to). This entropy value is calculated as shown below in Eqn. 6,

$$H = -\sum_{i=1}^{M} \frac{n_i}{N} \log_2(\frac{n_i}{N}) \tag{6}$$

where M is the number of total number of assignees in a project, N is the total number of tickets in a problem type (cluster) whose entropy you want to calculate, and n_i is the number of tickets from that cluster assigned to assignee number i. In experiment 3, we use this to demonstrate, using historical data, that clusters obtained by our PTC approach do exhibit low assignee entropy values and therefore are reflective of coherent problem types.

Our choice of the two new metrics introduced above are also driven by their complementary nature. While the *Textual Coherence* measures the clustering goodness based on the textual patterns, which is an *intrinsic* property of the tickets themselves, the *Assignment Entropy* measures the quality of clustering based on the steady-state practise observed in software engineering ecosystems and is therefore essentially an *extrinsic* measure. Therefore, employing both these metrics helps us comprehensively measure the goodness of the clusters obtained.

In both experiments 2 and 3, we compared the performance of our PTC approach against the following methods:

- **LDA:** Latent Dirichlet Allocation, which is the clustering technique of choice for most of the recent work in literature [10, 16, 19] employing text-based clustering. We used the LDA implementation from a tool called *MALLET*[3]. We configured MALLET according to the approach used in [19], where we keep the number of topics per subject as a constant (500 for our experiments) and associate a document j to a topic i only if document to topic proportion is >=0.25.

- **Random:** A random clustering approach where we create 1000 clusters per subject by randomly assigning tickets to each cluster. There are four cluster sizes per subject, selected as median of the 4 quartiles of cluster sizes in PTC.

- **Assignee-Only:** Clustering based on *Ticket Assignee* field, completely ignoring the problem text. Here we ignore individuals who have resolved less than 25 tickets.

Finally, in order to ensure that end users actually find the clusters useful, we did a user study where practitioners were asked to rate the clusters as meaningful i.e. all tickets in the cluster indicate the same problem that is repeatedly actioned. This evaluation is discussed in experiment 4.

[3]`http://mallet.cs.umass.edu/`

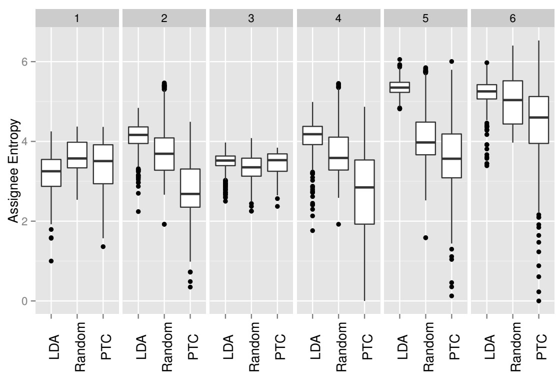

Figure 2: Assignee entropy of clusters for different techniques - LDA, Random and our PTC approach.

Experiment 1: Does PTC give high precision and recall when evaluated against a truth set?
Experiment 2: Does PTC give clusters with higher textual coherence when compared to other approaches?
Experiment 3: Does PTC give clusters with higher assignment entropy when compared to other approaches?
Experiment 4: Do end users find the clusters identified by PTC as coherent?

Experiment 1: (***Precision and Recall***) We observe that we obtain both high average precision (0.94) and recall (0.80) values. We also observe that the standard deviation values are quite low for both metrics (0.09 and 0.24 respectively) showing that the clustering is consistent across the various classes in the dataset. These values demonstrate that our approach produces good quality clusters on the small, hand-labeled dataset.

Experiment 2: (***Textual Coherence***) In this experiment we calculated the *Textual Coherence* scores obtained with our approach on each of the 6 IBM datasets and compared these scores against the ones obtained using LDA, Random and Assignee-Only. The results of this experiment are shown in the Fig. 1. The box plots show the mean and the variance of *Textual Coherence* scores for various clusters obtained using the four approaches across each of the 6 datasets. On comparing the distributions, we clearly observe that our approach yields clusters with higher *Textual Coherence* as compared to LDA and Random clustering. The textual coherence of our approach is also better than Assignee-only clusters, which validates the obvious that even though coherent problems are solved by a limited set of individuals, the same individual can solve multiple sets of coherent problems, and hence grouping tickets by assignee might not necessarily provide problem clusters of high *Textual Coherence*. This experiment reflects the better quality of clusters obtained from our approach as measured by their intrinsic similarity.

Experiment 3: (***Assignee Entropy***) Similar to the previous experiment, upon running the LDA and Random algorithm on each of the 6 datasets, we calculated the corresponding *Assignee Entropy* values and compared them against the ones obtained using PTC. We remind the reader that good clusters must exhibit low assignee entropy values whereas incoherent or dissimilar clusters exhibit high entropy. The results of this experiment are shown by the box plots in Fig. 2. As we can see, for Subject 1 and 3, we observe similar *Assignee Entropy* distributions between PTC, LDA and Random. However for each of the remaining 4 Subjects, we clearly observe

Table 5: Distribution of responses for Experiment 4

Total	Coherent	Not Coherent	Not Sure
92	59	28	5

that PTC yields clusters with lower (better) *Assignee Entropy* as compared to LDA. For these 4 subjects, the difference in assignee entropy between our and LDA approach was statistically significant with $p - value < 0.01$. Upon further investigation, we found in Subjects 1 and 3, the problem ticket assignment to people were more dictated by their availability rather than their specific skill. Some of the reasons for this are: 1) most of the tickets required a generic skills to fix or 2) the project had well documented resolution FAQs for reference.

Based on experiments 2 and 3, we observe that our approach yields clusters with better textual similarity and equivalent or better assignee entropy when compared to LDA, which is a popular topic identification technique. Our understanding is that this performance is primarily because we employ multiple clustering algorithms in PTC to handle the variance in the nature of text in problem tickets and merge their results to obtain representative clusters, therefore making it better-suited to this domain.

Experiment 4: (***User Study***) For this experiment, we first randomly picked problem clusters obtained from PTC across the range of assignee entropy and textual similarity scores to conduct a user-study on. We then contacted the technical experts from each of 6 IBM project teams and asked them the following question: *Are these problem clusters coherent enough to take unified responsive actions in your project?* We asked the experts to rate each problem cluster as either (1) coherent, (2) not coherent, or (3) not sure. Overall we had 92 problem clusters (data points) for the experts to verify across subjects. Per subject for each problem cluster, a pool of experts (3 to 5 people) provided their consolidated response.

From the response distribution seen in Table 5, we find that out of 92 responses (corresponding to 92 problem clusters), 59 (64%) clusters were marked as coherent by experts and 28 (30%) as not. We also observed that experts were able to identify the coherent categories without any deliberation, however for marking them as not coherent they took longer. This demonstrates that for a randomly selected subset of clusters, the experts found >50% coherent i.e. the problem cluster indicates a problem reported in multiple tickets in project and it could be removed by taking a single action.

These set of experiments indicate that our technique yields good clusters as measured using metrics of precision, recall, textual coherence and assignment entropy. In the user study, project experts further confirmed that a large percentage of identified clusters were useful.

4.2.2 RQ2: Efficacy of Individual Steps of PTC

Our approach had three steps: (1) noise removal, (2) clustering and (3) merging. Our clustering approach specifically focused on generating readable labels for clusters. In these sets of experiments we evaluate the efficacy of individual steps. The first step in our approach is to pre-process the tickets so as to remove noise. In experiment 5, we evaluate the impact of this step in the overall approach and measure it in terms of loss in precision and recall on the hand-labeled dataset when this step is removed from our PTC approach. The next step in our approach is to employ two different clustering algorithms and then combine the results using a merge algorithm. In experiment 6, we evaluate the effectiveness of using two algorithms and merging their results when compared to using only either of them. Again, we measure the effectiveness using the standard metrics of precision and recall on the hand-labeled dataset.

Table 6: Quantitative Evaluation of Improvements in the Cleansing and Merging Phases on Hand-labeled dataset.

	Before Cleansing		After Cleaning		After Merging	
	P	R	P	R	P	R
Avg	**0.66**	0.51	**0.92**	**0.54**	0.94	**0.80**
StdDev	**0.37**	0.38	**0.08**	0.23	0.09	0.24

Finally, we measure the effectiveness of our cluster labels with a user study in experiment 7 where practitioners from various projects were given the same cluster of tickets with two sets of labels. The first one were labels generated using our approach PTC and the other using the state of the art algorithm LDA. They were asked to qualitatively judge which label was more helpful from the point of view of comprehending the problem described by reading the label.

Experiment 5: What is the effectiveness of the noise removal step in PTC?
Experiment 6: Do we require multiple clustering methods and merging in PTC?
Experiment 7: Do end users find the labelling of clusters in PTC meaningful?

Experiment 5: (***Noise Removal***) Through this experiment, we evaluated the importance of the noise removal step of our approach by measuring the change in precision and recall values on the hand-labeled dataset as shown in Table. 6. We observed that before cleansing, the average precision was quite poor (0.66) implying that, on average, clusters had high contamination of tickets from classes other than the primary class. This was because clusters were being formed due to noisy, uninformative text across tickets. For e.g, a cluster was formed due to a common signature block of the same person across multiple classes. But, after cleansing, the average precision increased to 0.92 indicating that the clusters obtained contained tickets mostly from one class, along with a low value of standard deviation (0.08) indicating this to be the case across all classes. These improvements confirm the importance of the noise removal step of our approach. We used only the *Lingo* clustering algorithm for this experiment.

Experiment 6: (***Merge Clusters***) In this experiment, we compared the change in precision (P) and recall (R) values on the hand-labeled dataset when we just used one clustering algorithm (i.e. *Lingo*) versus running both algorithms and merging. As shown in Table. 6 a low value of average recall even after cleansing (0.54) indicated that not all of the 11 classes were being discovered by the clusters. But, after running both our clustering approaches and merging, we observed a rise in the recall to 0.80 showing that combining the results of multiple techniques helps increase the recovery of classes discovered by the clustering due to the ability to handle the variance in the nature of the text across these classes. This experiment demonstrates the importance of utilizing multiple clustering techniques and merging their results.

Experiment 7: (***User Study***) We conducted another user study to evaluate the effectiveness of labelling of the problem clusters and compared them against the ones obtained via LDA. To do so, we first identified overlapping clusters between PTC and LDA (clusters were marked as overlapping if they had 70% or more tickets in common). We then randomly picked 5 such overlapping clusters from each subject and presented them to the experts, asking them to answer the following question: *Is the label meaningful enough to describe the problem suggested by the cluster?* They could answer either (1) meaningful, (2) not meaningful, or (3) not sure. The experts were *not* made aware that these labels refer to overlapping 5 problem clusters, in order to allow them to make their judgements

Table 7: Distribution of responses for Experiment 7

Method	Total	Meaningful	Not Meaningful	Not Sure
LDA	25	6	17	2
PTC	25	19	6	0

about each label in isolation. Overall we had 60 data points for the experts to verify across subjects. We reached out to 6 experts for participation, but received responses from only 5 of them.

Table 7 presents the response distribution received. As we can see, out of the 25 overlapping problem categories between LDA and PTC, 17 of them were marked as not meaningful for labels generated through LDA while only 6 of them were marked not meaningful in PTC. 75% (19) of the problem categories labeled by PTC were marked meaningful, while on the other hand only 24% of LDA were marked as meaningful. This validates our claim that labelling the problem categories for easy comprehension is important towards reporting cohesive problem categories, and PTC provides more meaningful labels to experts as compared to LDA.

These set of experiments indicate that noise removal, use of multiple clustering algorithms and readable cluster labels are indeed essential steps in the approach and lead to significant improvements in the quality of output clusters.

4.2.3 RQ3: Usefulness of Clusters in IBM Projects

The quantitative and qualitative experiments performed above help us establish the goodness of our approach and necessity of individual steps. However, the actual benefit of the approach is realized only when project teams choose specific clusters identified using this approach and decide to implement a follow-up action which either identifies new requirements or reduces their maintenance cost. More than sixty different software maintenance projects within IBM have deployed our tool. We have several discussions with these practitioners to understand the kinds of actions project teams have been implementing using the clusters identified with our approach. In this section, we present two such case-studies. Since these are existing IBM customers, in the interest of confidentiality, we omit certain specific details but nonetheless report key observations and actions taken thereof.

Case Study 1: A particular project was observing a continuous increase in backlog of tickets. Both ticket volumes and average resolution time are key process metrics of concern to maintenance projects. Service Level Agreement (SLA) adherence could be adversely impacted if ticket volumes or ticket resolution times were to keep surging. To address these concerns, the project team primarily looked at those problem clusters which had a high volume of tickets in them. Amongst such clusters, of particular interest were those that took a longer time to fix. Below we discuss a few examples of unified actions this project implemented in response to these identified problem clusters.

Upon inspecting large clusters such as *cheque payment on LIV exception not working*, and *PO output ZNEU triggering for any field changed*, the teams realized that these problems point to design issues and can be avoided by implementing enhancements in the *payment* and *purchase order* processes respectively. Additionally, to address the problem of high resource turnover and to ensure that productivity is not lost when a new resources come in to solve tickets, the team started authoring FAQs for large problem clusters such as *Changing DSN Table Entries*, *Incorrect Instance in New User Forms*, *GSAP File Missing for Business Users*.

This project also found problem clusters that indicated a requirements drift. There was an existing functionality to convert a document to an OCR image.Once the image was successfully generated, it was transmitted to another system. The project kept receiving tickets thrice a week that required manual intervention to transmit the document. The identified problem clusters through PTC (such as *manually transmit document* and *OCR unsuccessful*) indicated that the OCR program was only able to handle documents that strictly adhered to a template. Based on this, the new requirement identified was to reject non-compliant documents upfront, even before they are passed to the OCR system.

Case Study 2: In another project, the software maintenance had been transferred over from another service provider to IBM. The new team did not have any prior knowledge of the repetitive problem patterns and therefore, used our approach to quickly come up to speed. The team observed that a large number of tickets formed clusters such as *password reset*. This was despite the fact that a self-help password-reset functionality was already available. A discussion with the client indicated that there was a user-awareness issue since the feature had only been recently implemented.

Further, the team found clusters such as *process chain failure*, *job abended*. These clusters were indicative of step failures in an otherwise large process chain. In most cases, the resolver would just re-initiate the process chain. Once the problem clusters were identified, it became easy for the IBM team to train there resolvers also to do the same. But this time, in response to the identified clusters, the team performed a deeper and more thorough investigation and found that many of these failures were reported on Fridays and resolved by Monday. It turned out that the system was shut down for routine maintenance for a couple of hours every Friday evening. Quite a few reports were also auto-configured to run at the same time. Now, as the system was not available during these times, the corresponding jobs as well as the process chains failed. The root cause of the problem was that over the years the knowledge and ownership of scheduled jobs and system maintenance had changed multiple hands and was now resident in multiple teams.

This issue led to a new requirement for the system maintenance team to change system downtime to a more opportune time. Also, a new application was developed to handle job failures during scheduled downtimes. System administrators advertised the system downtime and all problem tickets raised for job failures during that time period, were auto-queued for job restart once the system came up.

In general, we have found that PTC-found clusters helped project management teams better manage the volume of ticket streams and identify new requirements latent in these problem tickets. They are able to follow up with the technical leads to get a deeper understanding of the issues identified and decide on some unified actions. Building on this work, we are developing techniques to correlate these problem categories to operational (process) metrics such as volume, resolution time, SLA adherence to automatically identify those problem categories that cause majority of the operational grief and therefore present them in a prioritized manner.

4.3 Threats to Validity

Threats to external validity arise when the observed results cannot be generalized to other experimental setups. In our experiments we tried to limit this threat by evaluating on 80787 tickets across 6 different IBM client projects. The average number of words per ticket also varied across the subjects from 4 to 17. Also, our approach has been deployed within IBM and has been used by 60 different projects till date. We have received anecdotal evidence of its usefulness, some of which has been presented in *RQ3*. Finally, our approach has also proven to be useful in Solutioning (bidding for new projects), where the problem tickets are from client accounts which are not serviced by IBM.

Threats to internal validity arise when factors affect the dependent variables (the datasets described in Section 4.1). In our study, such factors are errors in implementation of cleansing and identifying candidate labels. Moreover, we used certain heuristics in our cleansing phase based on manual inspection of the tickets and guidance from project leads. The clustering approach required some configuration parameters such as minimum cluster size, minimum cluster merge threshold, minimum and maximum n-gram/phrase length etc. The choice of these parameters was heuristic based. Also factors like ticket assignment process, maturity of the project, geographical and cultural aspects of the people participating in the project could impact the *Assignee Entropy* metric.

The choice of metrics can be considered as threats to construct validity. Precision and recall are appropriate measures in search retrieval domain. However for measuring clustering efficiency of our approach in comparison with the golden set, we leverage these metrics. To address this, we used other metrics for evaluation and also performed user studies to validate our approach. Gold standard was produced by a single Subject Matter Expert (SME). While its true that in open domain (such as news articles, etc), there can be significant inter-annotator disagreement, in the domain of IT tickets where the notion of problem categories in operations are fairly well-understood and agreed upon, we did not see the need to calculate inter-annotator agreements. Further, since we use multiple clustering techniques which employ different measures, we wanted to employ a general *textual coherence* method for evaluation that is agnostic to all clustering methods compared. Also, we did not conduct a formal validation of the conjecture behind *Asignee Entropy* metric.

5. RELATED WORK

Using text-based analysis to identify groupings from software artifacts is not new. Most recently the work of [19] extracted topics from user queries on software forums. Having extracted the topics, they then try to create an FAQ (frequently asked question) from answers put in all the user queries grouped under a topic. In [10], authors have tried to extract automated labelled topics from commit-log comments recovered from source control systems to provide insights into evolving software development activities. Both these and other approaches such as [16] use LDA [4]. As we have seen in this paper, one of the primary challenges with using LDA is the labeling aspect of the extracted topics. Because of unreadable labels the topics are often discarded as not useful in our use case.

Prior work exist very similar to ours towards the specific goal of managing feature requests [5, 11]. In Frictionary [11], they proposed an approach to extract problem topics from Firefox support requests. They extract a problem topic from each request by transforming it into a predefined grammatical format (*subject-verb-object* such as *bookmark-disappeared-browser*) and by then aggregating the topics using certain heuristics. While such linguistic assumptions for transformation may be reasonable in their domain, they are highly restrictive for problem tickets, which, as shown before, can be machine-generated logs (no grammar), or email threads (contain too much noise). In [5], they focused on discussion threads on public forums and applied a known clustering algorithm called spherical K-means. Our approach builds upon these works by customizing known text clustering approaches to work well with noisy and diverse textual formats found in problem tickets seen in an industrial setting.

Text analysis approaches can be of two types: unsupervised such as IR-based techniques, clustering, topic modeling or supervised such as classification. Software engineering community has used for of these types to solve various use-cases. Multiple works [3, 21, 15] have applied unsupervised text analysis to detect similar or duplicate bug reports. Bug reports have more structure such as platform and components used, stack traces and free text. In [3], authors apply a different similarity detection approach for each of the constituents, for example 2 stack traces are similar if the method sequences being reported in trace are same, which 2 titles are similar if they contain same bag of words. [15], identify if a bug is duplicate by building using a combination of IR-based features and topic-based features. In [21] authors use natural language features along with execution information to find duplicate bug reports.

Supervised text analysis has been applied to propose solution to problems such as bug assignment [6], for classifying whether a given bug report is a feature or bug [1] or for determining whether a bug will be fixed or not [8]. In [6], authors learn a model per developer based on text similarity of maintenance requests and use it to assign these requests to an appropriate developer. In [1], authors train a naive bayes and regression based model to distinguish between bugs and enhancement requests. In [8], a regression model is trained on structured features available in bug reports such as who opened the bug, how many times it was re-assigned, who it was re-assigned to etc to determine if a bug will get fixed or not. A supervised approach would not work well in our scenario where problem categories are not known upfront and need to be discovered from problem tickets.

6. CONCLUSION

In this research, we addressed the problem of managing high-volume streams of trouble-ticket requests in an outsourced software maintenance ecosystem. Using unstructured information processing methods, we developed our “PTC" approach to cluster the stream into unified groups, and assigning suggestive, salient labels to the groups. We *quantitatvely* validated the clustering approach by showing that a) the groups were textually coherent, b) the groups were coherent enough to facilitate assignment to smaller teams drawn from the larger collection of project personnel, and c) our clustering approach was significantly better than commonly used LDA based approaches. A *qualitative* case study with domain experts suggested that our approach provided useful results which improved upon LDA based approaches. Our approach is quite general, and has proven to be useful in live, customer engagements on a wide range of software platforms (Oracle, Java, SAP, *etc*) and application domains; this experience suggests that this approach will be useful in a broad range of settings in the growing outsourced maintenance industry. In future work, we intend to explore beyond cluster size what other metrics we can use to rank/prioritize clusters for action by the account team. We also wish to explore how change in trends for high ticket volume problem categories are explainable through code changes being performed on application.

7. REFERENCES

[1] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc. Is it a bug or an enhancement?: A text-based approach to classify change requests. CASCON '08, pages 23:304–23:318. ACM, 2008.

[2] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 361–370. ACM, 2006.

[3] B. Ashok, J. Joy, H. Liang, S. K. Rajamani, G. Srinivasa, and V. Vangala. Debugadvisor: a recommender system for debugging. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC/FSE '09, pages 373–382. ACM, 2009.

[4] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[5] J. Cleland-Huang, H. Dumitru, C. Duan, and C. Castro-Herrera. Automated support for managing feature requests in open forums. *Communications of the ACM*, 52(10):68–74, 2009.

[6] G. A. Di Lucca, M. Di Penta, and S. Gradara. An approach to classify software maintenance requests. In *Proceedings. International Conference on Software Maintenance*, pages 93–102. IEEE, 2002.

[7] L. V. Galvis Carreño and K. Winbladh. Analysis of user comments: an approach for software requirements evolution. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 582–591. IEEE Press, 2013.

[8] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy. Characterizing and predicting which bugs get fixed: An empirical study of microsoft windows. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 495–504. ACM, 2010.

[9] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[10] A. Hindle, N. A. Ernst, M. W. Godfrey, and J. Mylopoulos. Automated topic naming to support cross-project analysis of software maintenance activities. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 163–172. ACM, 2011.

[11] A. Ko. Mining whining in support forums with frictionary. In *Proceedings of the 2012 ACM annual conference extended abstracts on Human Factors in Computing Systems Extended Abstracts*, pages 191–200. ACM, 2012.

[12] T. K. Landauer, P. W. Foltz, and D. Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[13] A. Mukhija and M. Glinz. Estimating software maintenance. *Lecture slides from Requirements*, 2003.

[14] G. Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

[15] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, pages 70–79. ACM, 2012.

[16] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 70–79. ACM, 2012.

[17] S. Osiriski, J. Stefanowski, and D. Weiss. Lingo: Search results clustering algorithm based on singular value decomposition. In *Intelligent information processing and web mining: proceedings of the International IIS: IIPWMÕ04 Conference held in Zakopane, Poland*, page 359, 2004.

[18] C. Pettey and R. van der Meulen. Gartner research - worldwide information technology outsourcing market report 2012. `http://www.gartner.com/newsroom/id/2021215`.

[19] S. Stefan HenSS, M. Monperrus, and M. Mezini. Semi-automatically extracting faqs to improve accessibility of software development knowledge. In *Proceedings of the 2012 International Conference on Software Engineering*, pages 793–803. IEEE Press, 2012.

[20] P.-N. Tan et al. *Introduction to data mining*. Pearson Education India, 2007.

[21] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th international conference on Software engineering*, pages 461–470. ACM, 2008.

[22] D. Weiss. *Descriptive clustering as a method for exploring text collections*. PhD thesis, Citeseer, 2006.

[23] D. H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.