2017 IEEE/ACM 39th International Conference on Software Engineering

Detecting User Story Information in Developer-Client Conversations to Generate Extractive Summaries

Paige Rodeghero, Siyuan Jiang, Ameer Armaly, and Collin McMillan
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN, USA
Email: {prodeghe, sjiang1, aarmaly, cmc}@nd.edu

***Abstract*—User stories are descriptions of functionality that a software user needs. They play an important role in determining which software requirements and bug fixes should be handled and in what order. Developers elicit user stories through meetings with customers. But user story elicitation is complex, and involves many passes to accommodate shifting and unclear customer needs. The result is that developers must take detailed notes during meetings or risk missing important information. Ideally, developers would be freed of the need to take notes themselves, and instead speak naturally with their customers. This paper is a step towards that ideal. We present a technique for automatically extracting information relevant to user stories from recorded conversations between customers and developers. We perform a qualitative study to demonstrate that user story information exists in these conversations in a sufficient quantity to extract automatically. From this, we found that roughly 10.2% of these conversations contained user story information. Then, we test our technique in a quantitative study to determine the degree to which our technique can extract user story information. In our experiment, our process obtained about 70.8% precision and 18.3% recall on the information.**

I. Introduction

A user story is a description of software functionality from the perspective of the software's user [1]. User story management is under the umbrella of requirements engineering, but a user story is different from a requirement in the traditional sense, because a story usually contains no information about how the software should be implemented. Instead, the story focuses on user experience, including the *role*, *function*, and *rationale* behind the user's objective in the format "as a ¡role¿, I want to ¡function¿, so that I can ¡rationale¿." For example, instead of a requirement "the system shall use a SQL database to store recently sold home prices", a user story might say "as a homebuyer, I want to search for recently sold homes, so I can estimate prices in my area."

Cohn [1] points out that the typical source of user stories is careful analysis of the conversations between programmers and customers. Since user stories do not contain implementation details, conversations with customers are an effective place to search for user stories. Developers are advised to take notes during conversations and reread these notes to write user stories by hand. This process is important because stories play a crucial role in Agile development, where release cycles are often short to accomodate the constantly-evolving needs of customers. The result is that developers are in a constant process of user story elicitation, as stories appear, mature, and are removed [2].

While there is an emphasis on by-hand effort for writing user stories in practice, software engineering literature describes automated summarization techniques for knowledge extraction from software artifacts. Notably, Rastkar *et al.* [3], [4] (ICSE 2010) built an algorithm for summarizing software artifacts, and tested the algorithm on a corpus of bug reports. That approach was based on earlier work by Murray and Carenini [5] designed to summarize emails and conversations. In a nutshell, these approaches are machine learning classifiers that are trained to recognize sentences in documents that are likely to contain certain types of information important to the summary. The performance was considered reasonable by human evaluators, at approximately 50% precision and 30% recall (see Section II-D for a more detailed discussion).

In this paper, we automatically extract data for writing user stories from records of conversations between developers and customers. Specifically, we 1) perform a qualitative study to test the hypothesis that conversations between developers and customers contain role, function, and rationale information for user stories, and 2) perform a quantitative study to determine the degree to which an existing classification algorithm can be trained to recognize this information in the conversations.

For the qualitative study (Section III), we recorded approximately 24 hours of spoken conversation (27 conversations total) between developers and customers over a period of three weeks at a software development company in the United States with between 30 and 50 employees. We then transcribed the recorded conversations to text, and manually annotated sections of the conversations as containing role, function, and/or rationale information pertaining to user stories. To ensure we could compare our results with earlier work, we also annotated "extractive summaries" for each conversation (see Section II-C for details on these summaries). We found that about 5.5% of the conversations included function information, 2.9% discussed rationale, but only 0.5% discussed role. About 10.2%

1558-1225/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE.2017.13

were part of the extractive summaries, which was slightly less than the 13% and 19% reported by Murray and Carenini [5] for the meeting and email corpora, respectively, and 28% reported by Rastkar *et al.* [3] for the bug reports. It is important to note that the Murray and Carenini meeting study, which is the most similar to our study since it also used annotated meeting transcripts, has the most similar extractive percentage. Put briefly, we found that the conversations included significant function and rationale information, but very limited data about the roles.

In the quantitative study (Section VI), we trained a classifier to recognize the function and rationale information, as well as the extractive summaries for comparision. We made numerous modifications from the technique described by Rastkar *et al.* and Murray and Carenini to adapt the technique to detect function and rationale data for user stories. We describe our procedure in detail in Section V. We obtained approximately 54.5% precision 24.0% recall for detecting sections of conversations containing function data, and 25.0% precision 26.9% recall for rationale data. For comparison purposes, we obtained about 70.8% precision 18.3% recall for extractive summaries.

Our long range vision for this research is to design an algorithm that generates user stories by listening to the conversations between developers and customers. The algorithm could, for example, be installed into teleconferencing software to automatically create notes about user stories for developers after a meeting. The technique we present in this paper is a research prototype in that direction; to be usable, it would need an automated transcriptionist and a natural language generation system to create "abstractive" user stories from the extractive data we currently can provide. Still, it is our view that this paper is a vital early step.

To facilitate reproducibility and assist other researchers, we have made our implementation available via an online appendix [1], including a virtual machine image with all dependencies installed. While we cannot release the recordings of the meetings for ethical and privacy reasons, we do provide our trained classifier so that it can be tested on other datasets. To our knowledge, no public records of meetings with user story information are available, but for comparision purposes, we duplicate our quantitative experiments for extractive summaries on the public AMI meeting corpus [6].

II. Background and Related Work

This section describes supporting technologies for this research, as well as related literature. Note that these technologies have been proposed and evaluated in previous work. We include them here because our work is based on these earlier techniques.

A. User Story Elicitation

User stories play a crucial role in Agile development, where programming activities are centered around the needs of customers. In a "textbook" Agile environment, developers elicit user stories through contact with customers, prioritize these stories, and schedule tasks in release cycles based on the stories. One characteristic of Agile development is a relatively short release cycle that is responsive to changing requirements [7]. The result is that developers are in a constant process of user story elicitation, as stories appear, mature, and are removed.

This process of elicitation is often messy: customers may have difficulty articulating their own needs, and developers may miss opportunities to ask clarifying questions or highlight important problems. Cohn [1] reinforces an opinion by Robertson and Robertson [2] that elicitation is akin to trawling for fish, as numerous passes are needed with different tools and techniques in order to catch as many user stories as possible. Nevertheless, Cohn [1] points out that a key component of user story elicitation is careful analysis of the conversations between programmers and customers. Developers are advised to take notes during conversations and reread these notes to write user stories by hand. Some researchers, such as Berenbach *et al.* [8], are even trying to introduce unique frameworks to better capture and analyze these elicitations.

B. Turn-based Conversation Analysis

In this paper, we use *turns* instead of *sentences* as the unit of analysis. This section defines these terms and outlines our motivation for using turns.

One important characteristic of the work by Rastkar *et al.* is that it is based on the *sentences* in the bug reports. The extractive summaries are a subset of the sentences in the bug reports. Likewise, the work by Murray and Carenini creates summaries from the sentences in emails and transcripts. For written text and some types of spoken text, sentence-based analysis is ideal because the boundaries between sentences are clear, and written sentences tend to contain cohesive information [9].

In contrast, the preferred unit of analysis for most types of spoken language is the *turn*, as noted by numerous authors in the field of conversation analysis in sociology [9], [10], [11], [12], [13]. A turn is the unit of speech that occurs when a person speaks in a conversation, between other speakers. In ordinary conversation, people take turns speaking and listening to others speak. Turns are different than sentences in that turns are dependent on the context in which the speaker takes a turn, and the speaker's own immediate thoughts and reactions. Human factors are present in spoken turns to a higher level than written sentences; social rank, confusion, number of listeners, etc., affect the length, order, and content of turns.

Transcriptionists are tasked with creating written sentences from spoken language. They will typically divide each turn into multiple sentences, marking punctuation including periods and commas. But this is a highly subjective process, since speakers may repeat or rephrase information, or fill gaps while thinking with "hmms" and "uhhs." The listeners in a conversation will typically defer to a speaker to finish a turn, even if the turn contains unfinished or run-on sentences.

[1] http://www3.nd.edu/~prodeghe/projects/userstories/

Our view is that role, function, and rationale information are much more likely to be detectable in turn-based analysis than sentence-based. The reason is that in a conversation, a customer may struggle to describe, for example, the function that he or she needs. The customer may describe a situation in many ways, splitting important information over several sentences or partial sentences. The developers will listen carefully during the customer's turn, to allow the customer to finish his or her thought. In these situations, it is far more useful to have the entire turn, rather than try to detect a single sentence which contains all the necessary information – such a sentence may not exist.

C. Summarization and Knowledge Extraction

Summarization is the process of creating a short description of a longer artifact [14], [15]. There are two types of summaries: extractive and abstractive. An extractive summary is a summary created out of pieces of the larger artifact. Sentence selection is a form of extractive summary generation, because it picks one sentence out of a document that describes the main points of the document [16]. A commonly-used extractive technique in software engineering research is a TF/IDF vector space model, in which the top *n* words are selected from a software artifact to describe that artifact [17]. In contrast, abstractive summaries are synthesized from information inside the artifact, without copying that information. Abstractive summarization is typically what humans do, as it often involves contextual information or "in your own words" interpretation. Nevertheless, abstractive summarization techniques do exist in the software engineering literature, such as work by McBurney *et al.* [18], Sridhara *et al.* [19], [20], [21], Moreno *et al* [22], [23], [24], and Buse and Weimer [25], [26]. These techniques are related to our work in that they use summarization techniques to extract knowledge from software artifacts, but differ widely in their approach, the type of data they are summarizing, and the type of artifacts they analyze. To our knowledge, we describe the first technique to automatically extract user story information from developer/customer meeting transcripts.

In this paper, we treat summarization as a form of knowledge extraction, in that a summarizer is attempting to extract and highlight a specific type of information from the larger artifact [27], [28]. We believe this is similar to, but unique from, the work done in Requirements Engineering by Cleland-Huang *et al.* [29] and others. In a nutshell, instead of summarizing a whole document, we aim to extract the parts of the document pertaining to the role, function, and rationale behind user stories. We apply summarization to the problem of knowledge extraction about user stories.

D. Summarizing Software Artifacts

Three key publications behind this research are by Murray and Carenini [5] in 2008, and a related advancement on that work by Rastkar *et al.* [3], [4] published in ICSE 2010 and TSE in 2014.

Murray and Carenini describe a technique to produce extractive summaries of email threads and meeting transcripts. The technique is essentially a machine learning classification problem, in which one class of sentences includes the sentences in the extractive summaries, and another class includes all other sentences. Generally speaking, their approach occurs over three steps: 1) compute quantifiable attributes for each sentence in the emails and meetings, 2) train a logistic regression classifier to detect sentences that are in manually-annotated extractive summaries, and 3) conduct a cross-validation experiment and compute performance metrics. The experiment they conducted was a "leave one conversation out" format, in which they trained on all conversations except one, and then tested on the remaining conversation. One major research contribution of their work is that the attributes they computed were generic in the sense that they did not depend on domain-specific terminology. The attributes can be calculated on multi-modal conversations instead of a specific type of conversation.

Rastkar *et al.* demonstrated how to apply Murray and Carenini's multi-modal conversation summarization to software artifacts. Using the same set of attributes, Rastkar automatically generated extractive summaries for bug reports. They manually annotated 36 bug reports by marking sentences in the bug reports that belonged to extractive summaries. They recruited three programmers to annotate each bug report, and then used a voting procedure to choose a "gold set" of annotations for each bug report. The performance of the classification was 57% precision and 35% recall. A user study with human experts found that this performance level was acceptable, with the experts rating 3.54/5.00 that the summaries represented the important points of the bug report.

III. Field Observations

We conducted a series of field observations to create a corpus of meeting records between developers and customers. This section describes our methodology for collecting these observations. This section also describes our qualitative analysis of the observations, including our research questions for the qualitative study and our annotation procedure.

A. Research Questions

Our research objective in this qualitative evaluation is to test the hypothesis that conversations between developers and customers contain role, function, and rationale information. This hypothesis stems from existing literature that emphasizes the importance of customer-developer conversations in user story elicitation [1], [2]. Specifically, it is important that we know whether our dataset contains this information to prepare for our quantitative analysis in Section VI. Therefore, we ask the following Research Questions (RQs):

RQ_1 What percent of the turns in the conversations contain role information, function information, and rationale information related to user stories?

RQ_2 What percent of the turns in the conversations belong to extractive summaries of those conversations?

TABLE I: A comparison of the size of our corpus to related work.

Related Experiment	Corpus	Speech	# of Conversations	# of Turns	# of Sentences
Murray and Carenini [5]	Enron Emails [30]	*n/a*	39	351	4600
Murray and Carenini [5]	AMI Meetings [6]	100 hours	138	64419	89302**
Rastkar *et al.* [3], [4]	Bug Reports	*n/a*	36	457	2361
This Paper	Devel/Customer Meetings	24 hours	27	3176	6303**

** Approximate. This is the count of sentences recorded by a transcriptionist. See Section II-B.

The purpose of RQ_1 is twofold. First, while it is widely accepted that the elicitation of user stories should result from these conversations [1], it is possible that not all of the information necessary to write a user story is contained in these conversations. If the information is not in the conversations, then it is not possible to automatically extract it, and there is no reason to study it in our quantitative analysis in Section VI. Second, that quantitative study depends on the turns being annotated as having role, function, and/or rationale information. Answering RQ_1 provides the opportunity to complete this annotation.

We ask RQ_2 in order to provide a mechanism for comparing the performance of our classifier (Section V) to previous work. It is not possible for us to release the records of conversations we collect, which means that we will need to use a public dataset to compare our classifier to earlier ones. However, our technique is designed for user story information, while the available public datasets have only extractive summaries annotated. Therefore, we annotate our dataset with extractive summaries to maintain a consistent baseline.

B. Methodology

This section describes our methodology for answering our research questions. We first describe how we created the corpus of meeting records. Second, we describe how we annotated that corpus with role, function, and rationale data, as well as extractive summaries.

C. Data Collection

The data that we collect includes recordings of meetings between developers and customers. To collect this data, the first author spent two months working as a software engineering intern at a software development company in the United States with between 30 and 50 employees (for privacy purposes, the company is anonymous). During this time, she was invited to regular stand up meetings and teleconferences with the client. These meetings consisted of discussions about progress on current tasks and plans for future tasks. After obtaining appropriate permission, the author observed, took notes, and recorded audio during each meeting. In total, nine meetings between developers and customers were recorded. We then hired a professional transcriptionist to create written transcripts of the audio for each meeting.

The transcripts contained an entire meeting's dialogue. However, each meeting contained several conversations, each of which had a separate topic with separate people (for instance, people joining or leaving a conference call as they are needed for each topic). Therefore, we manually separated each transcript into a set conversations. We understand that this is a subjective process, so we used the following criteria to minimize any bias during conversation separation: 1) a current speaker(s) leaves the conversation, 2) a new speaker(s) joins the conversation, or 3) a noticeable shift in topic, such as switching to a new bug. After this process was complete, we had 27 conversations total over the nine meetings. Our dataset is comparable in size to datasets for related experiments, as shown in Table I.

D. Annotation Process

All of the authors annotated the conversations. Each conversation was manually annotated by at least 3 authors. We restricted annotation to authors-only in order to maintain privacy of the speakers. The assignment of conversations to annotators was random, to ensure an unbiased assignment. The authors then annotated each of their conversations, but were careful not to discuss any annotations with each other to avoid introducing a bias. Every annotation used the following format:

File: the file name of the transcript
Conversation: an ID number for the conversation
Abstractive: an "in your own words" summary
Extractive: a list of turns summarizing the transcript
Role: a list of turns summarizing the role
Function: a list of turns summarizing the function
Rationale: a list of turns summarizing the rationale

Note that the annotations include an abstractive summary. We included this summary as an exercise to help the annotators understand the content of the conversation, but otherwise the abstractive summary was not used in our experiments.

Once annotations were complete, we combined them using a voting process to create a final goldset annotation for each conversation. The voting process was a simple majority: we selected turns for the goldset if those turns were selected by at least two of the three annotators for that conversation.

E. Threats to Validity

One threat to the validity of these observations is that much of the conversation was over the phone. This caused some of the conversation to be slightly garbled, meaning the transcriber to may have misheard some of the conversation. Although possibly causing some turns to be slightly wrong, in our view this threat is minimal, since we did not encounter evidence

that the errors would significantly change the meaning or context of the conversation. To minimize this threat, we hired a professional transcriptionist with experience handling difficult audio.

Another threat with these observations is the subjective nature of manual annotations. We attempt to mitigate this threat with the voting process of a majority consensus among the set of authors who annotated each conversation. With this consensus check, the final combined annotation is less likely to be biased. In addition, we compute the metric Pyramid Precision (see Section VI-C), which computes precision weighted for the number of votes for each turn.

Finally, one threat to validity is the source of the data we collected. It is possible that our dataset is not representative of "typical" meetings between developers and customers. We aim to mitigate this threat by recording actual meetings at an active software development company (unlike the AMI meeting corpus, which is simulated). Still, it is possible that our partner company is different from other companies in a way that would affect our results.

IV. Field Observations Results

In this section, we present our answer to each research question, as well as our rationale, and interpretation of the answers. These answers are the basis for the quantitative study discussed later in the paper.

A. RQ_1: Turn Information

We found that out of the 3176 total turns, 5.5% of turns contained function information, 2.9% of turns contained rationale information, and only 0.5% of turns contained role information (see Figure 1). Our perception of why role information was missing from conversations is that during these regular meetings, the employees already had a understanding of their roles within these projects. Therefore, there was no need to identify each participant's role during the meeting in order for the speakers to complete their assigned tasks. In contrast, function and rationale information is present since it is usually unknown before the task is discussed in the meeting. Therefore, we only used function and rationale information during the quantitative study.

B. RQ_2: Extractive Summaries

For extractive summaries produced from our annotations, we found that 10.2% of turns belong to extractive summaries of their respective conversations (see Figure 1). For comparison, Murray and Carenini found 13.0% of their turns to be included within the extractive summaries for the AMI meetings and 19.0% of their sentences included in the Enron email summaries [5]. Also, Rastkar *et al.* found 28.3% of the bug report turns were included in the extractive summaries [3], [4]. In our view, our findings are consistent with the findings from the AMI meeting conversations.

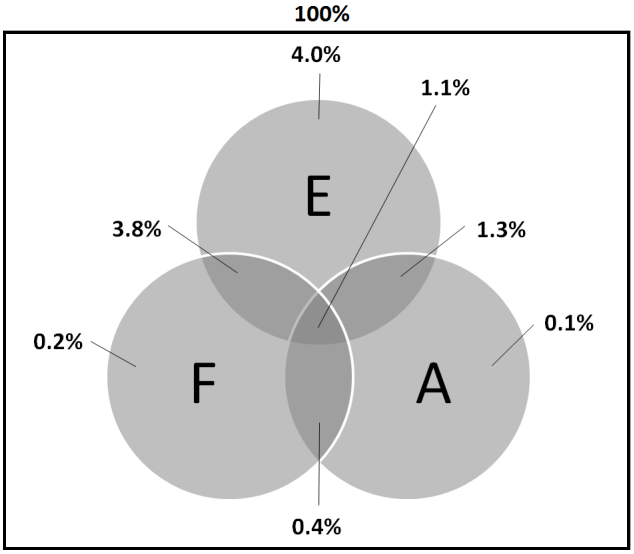


Fig. 1: Venn diagram of the percentages of turns that are included in each category. The categories are function (F), rationale (A), and extractive Summary (E). For example, 0.2% of turns were marked as including function information only, while 3.8% of turns had both function and extractive summary information only.

V. Our Approach

Our approach, is essentially a machine learning classifier, in which we classify turns in speech as either having user story information, or not. We build two classifiers for each type of data that we extract: two for function information, and two for rationale information. The two classifiers are based on two different algorithms. We also build two classifiers to create extractive summaries of the conversations, for purposes of comparison to related work. Note that because of the very small amount of role information in our dataset (see Section IV-A), we do not attempt to extract that type of information. Our approach is inspired by previous work (see Section II-D), but has numerous differences that we describe in this section.

A. Data Preparation

Any technique using supervised machine learning will depend on prepared data in the form of labels for the items to be classified. In our case, we labeled every turn in every conversation as containing role, function, and/or rationale data, as well as whether the turn belongs to an extractive summary. We completed this annotation process as part of our Field Observations in Section III. We use the same annotations in this section. This annotation process is depicted in Figure 2: we describe our process of obtaining the transcripts in area 1 in Section III-C, and the annotations in area 2 in Section III-D.

B. Attributes

We use a set of 25 attributes in each of classifier that we build (Figure 2, area 3). Table II provides a brief description

of each attribute. These attributes are in general similar to the attributes proposed by Murray and Carenini [5] and later used by Rastkar *et al.* [3], [4], and due to space limitations we refer readers to those publications for complete details. However, a key distinction is that the unit of analysis in our work is a turn, not a sentence, as described in Section II-B. A few of the attributes are identical, such as `spau`, which is the time between the current turn and the following turn. Also, a few attributes, such as `tloc` (the position of a sentence in its turn) are nonsensical when calculated on turns instead of sentences. But most attributes could be modified slightly, for example `thisent`. In Murray and Carenini's paper, `thisent` is the entropy of the current sentence. In our work, it is the entropy of the current turn.

From [5], these classification attributes fall into four categories: length, structural, participants, and lexical. Our attributes still fit into these categories. `Slen` and `wc` are in the length category; `cloc`, `ppau`, `spau`, `tpos1`, and `tpos2` are in the structural category; and `begauth` and `dom` are in the participant category. The remaining attributes, which are based on probability and entropy of textual data, are contained within the lexical category.

We have added two new attributes: `pent_empt` and `sent_empt`. These are when two of the entropy-based attributes, pent and sent, equal zero. This means that either all previous turns or all subsequent turns, respectively, have no entropy at all. These attributes are simpler descriptors of the turns than their numeric counterparts, which may provide better classifications for sparser data. These new attributes still fall under the lexical category.

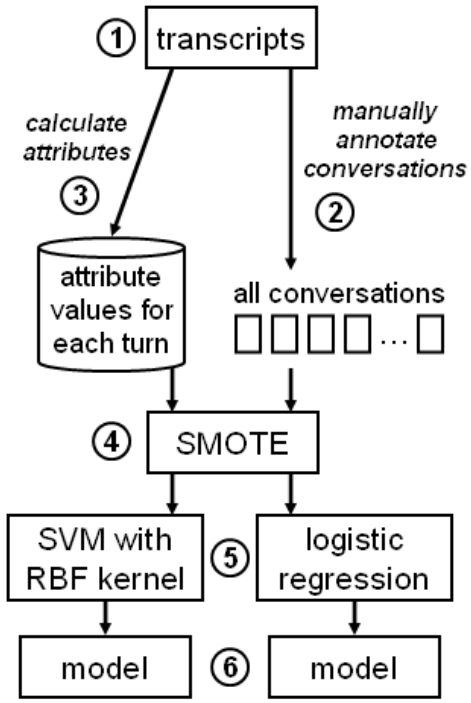


Fig. 2: Overview of our approach. We completed steps 1 and 2 while answering RQ_1 and RQ_2 ins Section III. Steps 3, 4, 5, and 6 are described in Section V.

TABLE II: List of attributes we calculate.

Attribute	Description
begauth	first participant in convo
cent1	cos. of turn and convo., w/ Sprob
cent2	cos. of turn and convo., w/ Tprob
cloc	position in convo.
cos1	cos. of convo. splits, w/ Sprob
cos2	cos. of convo. splits, w/ Tprob
cws	rough ClueWordScore
dom	participiant dominance in words
mns	mean Sprob score
mnt	mean Tprob score
mxs	max Sprob score
mxt	max Tprob score
pent	entropy of convo. before the turn
pent_empt	no entropy in convo. before the turn
ppau	time btwn. current and prior turn
sent	entropy of convo. after the turn
sent_empt	no entropy in convo. after the turn
slen	word count in turn, globally normalized
sms	sum of Sprob scores
smt	sum of Tprob scores
spau	time btwn. current and next turn
thisent	entropy of current turn
tpos1	time from beg. of convo. to turn
tpos2	time from turn to end of convo.
wc	word count in turn, not normalized

Prior to training with each algorithm, we scaled the attribute values to ensure they would be between 0 and 1 to avoid some attributes from dominating the others [31].

C. Adaption for Low Incidence Data

One challenge with our dataset is that the incidence of function and rationale information is relatively low. In Section IV-A, we found that function data was present in 5.5% of turns and rationale data in 2.9% – the turns in the groups labeled as having function and/or rationale data are a small minority class. A common problem in supervised machine learning of low incidence data is that the algorithm may predict that everything is in the large class, while ignoring the minority class [32].

We used the SMOTE [33] algorithm to address this problem (Figure 2, area 4). SMOTE works by oversampling the minority class by creating synthetic examples of the minority class, and adding those examples to the dataset until the minority class is equal in size to the larger class. SMOTE has been shown to have generally good performance, outperforming duplicative oversampling as well as undersampling of the majority class for many datasets [33], [32].

D. Creating the Prediction Models

We built two prediction models for each of the types of information we extract: two for function data, two for rationale data, and two for extractive summaries. One model is based on the algorithm Support Vector Machines (with an RBF kernel) [34], and the other model is based on the algorithm Logistic Regression [35] (Figure 2, area 5). We used Logistic

Regression for two reasons: first, it is the algorithm that Murray and Carenini [5] and Rastkar *et al.* [3], [4] used to obtain reasonable performance. Second, Logistic Regression is a probabilistic classifier, which means that it returns a probability that each turn belongs to a class. The advantage to these probabilities is that the size of the predicted class can be set with a cutpoint; the top *n* turns can be selected. As a contrast, we also used the SVM-RBF algorithm, which is a prominent binary classifier. In Section VI, we compare the performance of these algorithms on our dataset.

E. Reproducibility and Implementation

To facilitate reproducibility and to assist other researchers, we release our prediction models as well as our complete implementation via an online appendix [2]. Our implementation is built using Scikit-learn [36], as well as custom scripts to parse the transcripts and calculate the attributes. These are all available online, along with a virtual machine image with all dependencies installed to demonstrate how it is used.

We also release the prediction models that we created as part of the procedure depicted in Figure 2 (area 6). Note that these are models trained on the entire corpus of conversations in our dataset – they are not the models we use for testing in the cross-validation experiments in the next section. We release these models in lieu of the transcripts, since we cannot release the transcripts for privacy reasons. Future researchers can use these models on their own datasets, similar to how Rastkar *et al.* tested a model on their own dataset that was created by Murray and Carenini [3].

VI. Cross-Validation Experiment

This section describes our quantitative study, which is a cross-validation experiment to evaluate our approach. We cover our research questions, our methodology and metrics for answering those questions, and threats to validity.

A. Research Questions

Our objective with this quantitative study is to determine the degree to which the classifiers we train are able to extract function and rationale information, as well as extractive summaries. As in Section V, we do not include role information because our dataset contains so little of it (see Section IV-A). We pose the following two questions:

RQ_3 What is the performance of the best-performing configuration of our approach, in terms of the metrics in Section VI-C?

RQ_4 Which attributes are the most informative for the classification task?

We ask RQ_3 because there are several potential configurations of our approach, and we seek the highest performing configuration. The configuration is the algorithm (SVM-RBF vs Logistic Regression) and the classification threshold for Logistic Regression. We ask RQ_4 for a similar reason. We use 25 attributes in our approach, and several of these we adapted for turn-based analysis instead of sentence-based. Some attributes may be more useful for classification than others. It is beneficial for us to report the usefulness of each attribute because it may be possible for future researchers to simplify the approach by removing some less useful attributes, without significantly harming performance.

[2]http://www3.nd.edu/~prodeghe/projects/userstories/

B. Methodology

The methodology we follow is depicted in Figure 3. Generally speaking, we follow a "leave one conversation out" procedure. The typical cross-validation process is either a leave-one-out or an *n*-fold validation. Leave-one-out usually means train all items in the dataset, then test on one. In an *n*-fold validation, 1/*n* items are randomly selected as the testing set. In our case, the items in the dataset are turns. In our view, the typical leave-one-out and *n*-fold validation do not reflect realistic scenarios, since the turns in one conversation almost never have any direct affect on the turns in another conversation. The realistic scenario is that a researcher has *m* conversations, and trains a classifier on those conversations. Then the researcher receives a "new" conversation, and classifies that conversation.

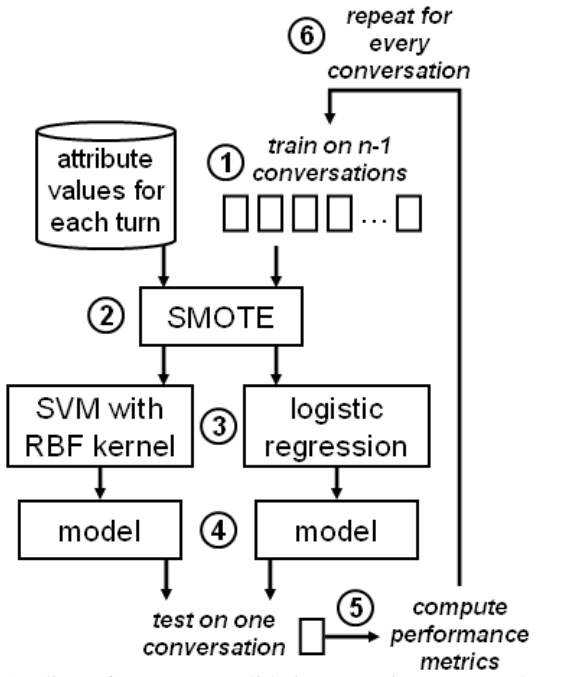


Fig. 3: Outline of our cross-validation experiment procedure. Note that the procedure is slightly different than the creation of the prediction models shown in Figure 2. In the experiment, we create a new model for each cross validation fold. In each fold, we train on all conversations except one, and then test on the one remaining conversation. With 27 conversations, our experiment has 27 folds.

To emulate that realistic scenario, we use a leave one conversation out process in which we set aside one conversation as the test set, and create a training set using the remaining conversations (Figure 3, area 1). Note that we use the SMOTE procedure (Section V-C) on the training set only (area 2) to avoid biasing the experiment – SMOTE changes the dataset, and a bias could occur if we modify testing data.

Next, we train both SVM-RBF and Logistic Regression algorithms (area 3). We list them together in Figure 3 to emphasize that we train both, but the algorithms are independent. They do not share data, and we do not attempt to combine the models that they create (area 4). We use each model to test the conversation that we left out of the training set, and compute our performance metrics during that test (area 5). We repeat this process for every conversation.

C. Metrics

For RQ_3, we calculate the standard machine learning performance metrics Precision, Recall, True Positive Rate (TPR), and False Positive Rate (FPR) [37], [32]. We also compute Pyramid Precision, which is similar to precision, except that it considers the number of human annotators who marked each turn. In Pyramid Precision, a prediction model is rewarded for predicting turns that have been annotated by more people as belonging to a class. Pyramid Precision is useful in cases where a goldset is created by multiple people who may disagree [38], and was used by Murray and Carenini [5] and Rastkar *et al.* [3].

Note that Logistic Regression generates predictions based on a probability threshold: turns with probabilities above the threshold are predicted as part of the class. That is in contrast to SVM-RBF, which provides binary predictions (see Section V-D). Therefore, for SVM-RBF, we report the average of the performance metrics over all folds of the cross validation. But for Logistic Regression, we report the optimal probability threshold.

For RQ_4, we calculate the standard metric F-score [39] for each attribute. F-score is commonly used in feature selection to determine which attributes are the most informative for a classification task. We report F-score for both SVM-RBF and Logistic Regression.

D. Threats to Validity

One threat to validity to this study is our data source, as mentioned in Section III-E. There is a possibility that the data set we use is not representative of "typical" developer-customer meetings. However, this threat was mitigated by recording several real meetings taking place at an active software development company.

Another threat to validity exists in the selection of attributes. Although it is a relatively large set and most were used in previous studies, they may not represent all usable attributes for classification tasks. In addition, the conversion of certain attributes from sentence-based to turn-based may have affected the usefulness of those attributes. This is handled, however, by the modeling itself and the analysis performed in Section VII-B.

VII. Cross-Validation Results

In this section, we present our answer to each research question, as well as our rationale, and interpretation of the answers.

A. RQ_3: Best Configuration

We found the best configuration for classification is Logistic Regression (92% threshold) (see Table II), though the performance of the algorithms is similar. Logistic Regression outperforms SVM-RBF with the more traditional metrics of Precision, Recall, True Positive Rate (TPR), and False Positive Rate (FPR). The TPR and FPR are represented together as the Receiver Operating Characteristic Area Under the Curve (AUROC) score. However, with the Pyramid Precision (PP) metric, which takes the number of annotators into account, SVM-RBF performs better with two of the three types of information.

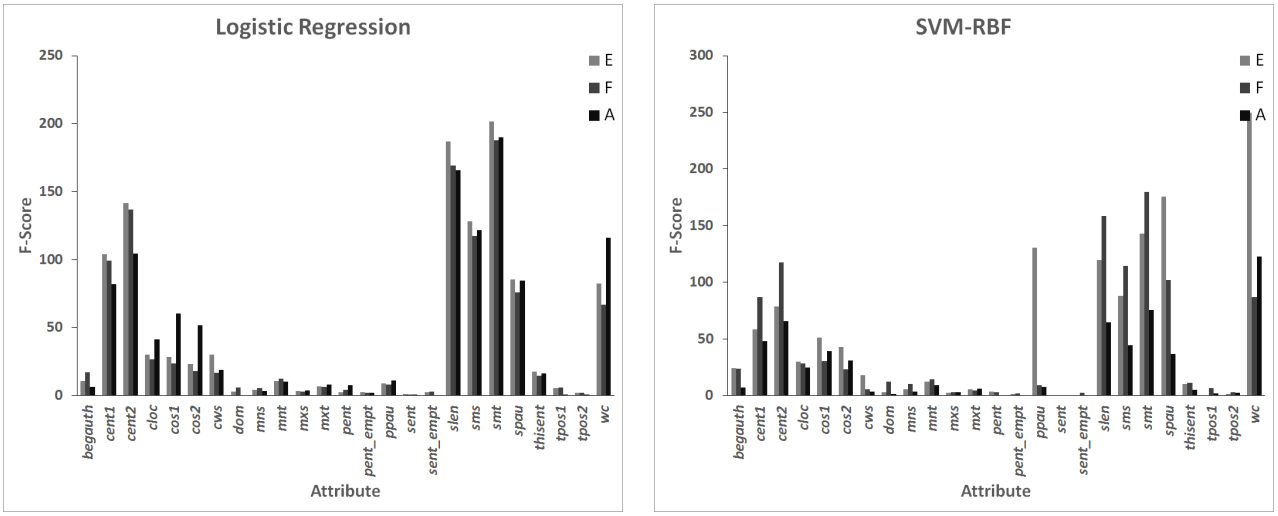
TABLE III: Metric results using our corpus. Details shown are Type, Classifier & Threshold where applicable (Class.), Precision (Prec.), Recall(Rec.), Pyramid Precision (PP), and AUROC score.

Type	Class.	Prec.	Rec.	PP	AUROC
E	LR(92%)	0.708	0.183	0.597	0.587
E	SVM	0.684	0.140	0.649	0.566
F	LR(94%)	0.545	0.240	0.472	0.614
F	SVM	0.522	0.240	0.491	0.613
A	LR(81%)	0.250	0.269	0.364	0.622
A	SVM	0.182	0.154	0.204	0.566

With the extractive summary (E) information, the Logistic Regression model performed the best using traditional metrics, as mentioned above, with a Precision of 70.8%, a Recall of 18.3%, and a AUROC score of 0.587. In terms of Pyramid Precision, the SVM model with RBF kernel outperformed Logistic Regression with a PP of 64.9%. Although the PP was higher for SVM-RBF, it still did not overcome the Logistic Regression's Precision, which we believe means that Logistic Regression outperformed SVM-RBF overall.

We observe a similar pattern with function (F) information. The Logistic Regression model performed the best using traditional metrics with a Precision of 54.5%, a Recall of 24.0%, and a AUROC score of 0.614. In terms of Pyramid Precision, the SVM model with RBF kernel outperformed Logistic Regression with a PP of 49.1%, though the results are extremely close. Logistic Regression outperforms SVM-RBF overall, though the margin is not large enough to justify a strong recommendation either way.

With the rationale (A) information as the type, the Logistic Regression model outperforms the SVM model for all metrics. This configuration had a Precision of 25.0%, a Recall of 26.9%, a PP of 36.4%, and a AUROC of 0.622. With this configuration, there are a couple differences to note other than Logistic Regression having a better PP than SVM-RBF. One observation is that this is the only configuration where the PP measures higher than the Precision. Another observation



(a) Logistic Regression (92% threshold)

(b) SVM-RBF

Fig. 4: F-scores of the 25 attributes after running both models on our corpus. (a) includes the Logistic Regression model using extractive summary (E), function (F), or rationale (A) information. (b) is the same for the SVM model using the RBF kernel.

is that the ROC score is highest with this A information than with the E and F information. We believe all three of these observations are due to the lower incidence of rationale information compared to extractive summary and function information. With less data to use, the traditional prediction becomes more difficult, but the metrics that make use of other data that balances them become more useful.

From all these configurations, the best overall configuration for the classification task is the Logistic Regression model, especially for the extractive summary information. As mentioned in Section VI-C, Logistic Regression uses a probability threshold to determine classification. This configuration uses a probability threshold of about 0.92, which produces a lower FPR at the cost of a lower TPR.

In terms of the most-closely related work (Section II-D), we caution that our data is different (transcripts vs. bug reports and emails), and that our analysis is turn-based rather than sentence-based. Still, the previous results provide a rough basis for comparision (a "sanity check"). The Murray and Carenini AMI meeting experiments produced a best case Pyramid Precision of 23.0% and AUROC score of 0.850 [5]. The Murray and Carenini Email experiments produced a best case Pyramid Precision of 47.0% and AUROC score of 0.740 [5]. The Murray and Carenini studies did not report Precision and Recall, but they did mention that they recorded high Precision and low Recall values. The Rastkar Bug Report experiments produced a best case Pyramid Precision of 63.0%, an AUROC score of 0.722, a Precision of 57.0%, and a Recall of 35.0% [3], [4]. While these numbers are not directly comparable, we do note that high precision and low recall is a common feature of the approaches, and that Rastkar *et al.* reported that this was acceptable during a study with human experts [4].

B. RQ_4: Best Attributes

We found 7 of the 25 attributes to consistently be the most informative for this classification task across both models and all three types of turn information. As shown in Figure 4, these attributes tended towards higher F-scores (see Section VI-C) compared to other attributes. These attributes are `cent1`, `cent2`, `slen`, `sms`, `smt`, `spau`, and `wc` and are briefly described in Table II. As explained in Section V-B, these attributes can be separated into four distinct categories: length, structural, participants, and lexical. Of these 7 most informative attributes, 2 are in length, 1 is in structural, 0 are in participants, and 4 are in lexical. It is important to note that attributes within the lexical category perform the best in this classification task, which we believe is because they are created using the most information. Another important observation is that the attributes representing participant specifics have little to no effect on the classification with our data. We believe that is because the speech in each conversation is so broken between speakers that it is difficult for the algorithm to use it effectively. We also found 4 somewhat informative attributes: cloc (structural), cos1 (lexical), cos2 (lexical), and cws (lexical). As can be seen in Figure 4, these 4 attributes do not get as high F-scores as often as the other 7, but they do occasionally score higher than the strictly non-informative attributes.

Among all the configurations, the one containing the highest F-scores for the informative attributes is the Logistic Regression model with the extractive summary information. This is the same configuration shown in Section VII-A.

It is our view that our most informative set is comparable to the sets produced by the previous studies mentioned in Section II-D. All three experiments agree that `cent1`, `cent2`, `cws`, `slen`, `sms`, and `smt` are informative for classifica-

tion. The Murray and Carenini AMI experiments and Email experiments additionally have the `mxs` and `mxt` attributes included. Also, the Murray and Carenini AMI experiments and the Rastkar Bug Report experiments list the `slen2` attribute as informative. Compared to our set, almost everyting is in agreement. We do not include `cws`, `slen2`, `mxs`, or `mxt`. However, `cws` was somewhat informative with our approach, and `slen2` was not included in our usable attributes to begin with because of our use of the turn/conversation system.

VIII. AMI Experiments

As mentioned in Section I, we duplicated our quantitative experiments (Section VI) for extractive summaries on the public AMI meeting corpus [6]. Since we cannot ethically release any of our meeting recordings from the active software company, these experiments on the AMI corpus serve as a reproducible example of our work. Although the AMI dataset uses artificially created meetings and conversations, it is a popular dataset due to its size and depth of information. For these experiments, we followed the same approach as outlined in Figure 3, except that there were many more folds since the API corpus contains more conversations. However, since the AMI dataset only includes extractive summaries, we do not use function and rationale information for these experiments. We have made our implementation and this example available via an online appendix (see Section V-E).

Using the same metrics we used in our quantitative study (see Section VI-C), we found the metric values to be more split between the two models, as can be seen in Table IV. For two of the metrics, Logistic Regression outperforms SVM-RBF with a Recall of 24.9% and an AUROC score of 0.619. The Receiver Operating Characteristic Area Under the Curve (AUROC) score is a combined representation of the True Positive Rate (TPR) and the False Positive Rate (FPR). For the other two metrics, the SVM model with the RBF kernel outperforms the Logistic Regression model with a Precision of 78.2% and a Pyramid Precision of 82.6%. In general, both algorithms had generally similar performance, so from our view, future researchers may choose either algorithm depending on other factors affecting their experiments. Also, we observed that the same 7 attributes that were found to be most informative in our study were most informative in terms of F-score: `cent1`, `cent2`, `slen`, `sms`, `smt`, `spau`, and `wc`.

One may observe that our AMI experiments produced a best case Pyramid Precision of 82.6% and AUROC score of 0.619, and the Murray and Carenini AMI experiments produced a best case Pyramid Precision of 23.0% and AUROC score of 0.850. As in Section VII, we caution that these numbers are *not* comparable since the unit of analysis is different (turns vs. sentences). We reiterate that we include this section only to observe trends and to provide a reproducible baseline for our approach. Our intent is that future researchers in the area of summarization and user story generation can build and test their approach on this reproducible baseline of public data, to verify that their approach functions similarly to ours.

TABLE IV: Metric results using the AMI meetings corpus. Details shown are Type, Classifier (Class.), Precision (Prec.), Recall(Rec.), Pyramid Precision (PP), and AUROC score.

Type	Class.	Prec.	Rec.	PP	AUROC
E	LR(97%)	0.758	0.249	0.791	0.619
E	SVM	0.782	0.158	0.826	0.576

IX. Discussion and Conclusion

In this paper, we advance the state of the art in two key directions. First, we contribute the analysis of actual user story conversations to the software engineering literature. Although we cannot publish the conversations themselves due to privacy concerns, previous work in user story analysis used the AMI meeting corpus, which was artificially created. In our qualitative study (Section IV), we found that about 5.5% of the turns included function information, 2.9% discussed rationale, but only 0.5% discussed role. About 10.2% were part of the extractive summaries. From these results, we have two key findings: 1) while function and rationale information are available for analysis in real conversations, role information is not; and 2) in terms of extractive summary information, the artificial AMI dataset is a comparably useful source of information as an active meeting dataset.

Second, we create a novel approach that extracts user story information from transcripts of spoken conversations. In our quantitative study (Section VII), we obtained approximately 54.5% precision and 24.0% recall for detecting sections of conversations containing function data, and 25.0% precision and 26.9% recall for rationale data. For comparison purposes, we obtained about 70.8% precision and 18.3% recall for extractive summaries. Compared to similar previous studies, we found our results to be comparable in most cases and better in others, most notably with the Pyramid Precision metric. With these results, this new approach provides an extra option for researchers if they feel their data is either more appropriately broken into turns or simply cannot be represented as full sentences.

Acknowledgment

We strongly thank Karen Hooge Michalka and Dr. Lyn Spillman from the University of Notre Dame Department of Sociology for their advice in preparing the qualitative study in this paper. We also thank teacher Michael Dimino of the Concord Public School district for participating in portions of this research. We hugely appreciate the active software company for allowing the recording of several of their customer meetings, as well as the many employees who allowed the collection of their data. This work was partially supported by the NSF grants CCF-1452959 and DGE-1313583, and the Office of Naval Research grant N000141410037. Any opinions, findings, and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

REFERENCES

[1] M. Cohn, *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.

[2] S. Robertson and J. Robertson, *Mastering the requirements process: Getting requirements right*. Addison-wesley, 2012.

[3] S. Rastkar, G. C. Murphy, and G. Murray, "Summarizing software artifacts: a case study of bug reports," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 2010, pp. 505–514.

[4] ——, "Automatic summarization of bug reports," *IEEE Transactions on Software Engineering*, vol. 40, no. 4, pp. 366–380, 2014.

[5] G. Murray and G. Carenini, "Summarizing spoken and written conversations," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2008, pp. 773–782.

[6] I. McCowan, J. Carletta, W. Kraaij, S. Ashby, S. Bourban, M. Flynn, M. Guillemot, T. Hain, J. Kadlec, V. Karaiskos *et al.*, "The ami meeting corpus," in *Proceedings of the 5th International Conference on Methods and Techniques in Behavioral Research*, vol. 88, 2005.

[7] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries *et al.*, "Manifesto for agile software development," 2001.

[8] M. Gall and B. Berenbach, "Towards a framework for real time requirements elicitation," in *2006 First International Workshop on Multimedia Requirements Engineering (MERE'06 - RE'06 Workshop)*, Sept 2006, pp. 4–4.

[9] P. Ten Have, *Doing conversation analysis*. Sage, 2007.

[10] C. E. Ford, B. A. Fox, and S. A. Thompson, *The language of turn and sequence*. Oxford University Press on Demand, 2002.

[11] I. Hutchby and R. Wooffitt, *Conversation analysis*. Polity, 2008.

[12] T. Nishida, *Conversational Informatics: An Engineering Approach*. Wiley, 2007.

[13] E. A. Schegloff, *Sequence Organization in Interaction: A Primer in Conversation Analysis I*. Cambridge University Press, 2007.

[14] R. Barzilay, K. R. McKeown, and M. Elhadad, "Information fusion in the context of multi-document summarization," in *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*, ser. ACL '99. Stroudsburg, PA, USA: Association for Computational Linguistics, 1999, pp. 550–557. [Online]. Available: http://dx.doi.org/10.3115/1034678.1034760

[15] V. Gupta and G. S. Lehal, "A survey of text summarization extractive techniques," *Journal of emerging technologies in web intelligence*, vol. 2, no. 3, pp. 258–268, 2010.

[16] J. Goldstein, M. Kantrowitz, V. Mittal, and J. Carbonell, "Summarizing text documents: sentence selection and evaluation metrics," in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 1999, pp. 121–128.

[17] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the use of automated text summarization techniques for summarizing source code," in *2010 17th Working Conference on Reverse Engineering*. IEEE, 2010, pp. 35–44.

[18] P. W. McBurney and C. McMillan, "Automatic source code summarization of context for java methods," *IEEE Transactions on Software Engineering*, vol. 42, no. 2, pp. 103–119, 2016.

[19] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, "Towards automatically generating summary comments for java methods," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ser. ASE '10. New York, NY, USA: ACM, 2010, pp. 43–52. [Online]. Available: http://doi.acm.org/10.1145/1858996.1859006

[20] G. Sridhara, L. Pollock, and K. Vijay-Shanker, "Automatically detecting and describing high level actions within methods," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 101–110. [Online]. Available: http://doi.acm.org/10.1145/1985793.1985808

[21] ——, "Generating parameter comments and integrating with method summaries," in *Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension*, ser. ICPC '11, 2011, pp. 71–80. [Online]. Available: http://dx.doi.org/10.1109/ICPC.2011.28

[22] L. Moreno, "Summarization of complex software artifacts," in *Companion Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE Companion 2014. New York, NY, USA: ACM, 2014, pp. 654–657. [Online]. Available: http://doi.acm.org/10.1145/2591062.2591096

[23] L. Moreno and J. Aponte, "On the analysis of human and automatic summaries of source code," *CLEI Electronic Journal*, vol. 15, no. 2, pp. 2–2, 2012.

[24] L. Moreno, J. Aponte, S. Giriprasad, A. Marcus, L. Pollock, and K. Vijay-Shanker, "Automatic generation of natural language summaries for java classes," in *Proceedings of the 21st International Conference on Program Comprehension*, ser. ICPC '13, 2013.

[25] R. P. Buse and W. R. Weimer, "Automatic documentation inference for exceptions," in *Proceedings of the 2008 international symposium on Software testing and analysis*, ser. ISSTA '08. New York, NY, USA: ACM, 2008, pp. 273–282. [Online]. Available: http://doi.acm.org/10.1145/1390630.1390664

[26] ——, "A metric for software readability," in *Proceedings of the 2008 International Symposium on Software Testing and Analysis*, ser. ISSTA '08. New York, NY, USA: ACM, 2008, pp. 121–130. [Online]. Available: http://doi.acm.org/10.1145/1390630.1390647

[27] D. R. Radev, E. Hovy, and K. McKeown, "Introduction to the special issue on summarization," *Computational linguistics*, vol. 28, no. 4, pp. 399–408, 2002.

[28] L. F. Rau, P. S. Jacobs, and U. Zernik, "Information extraction and text summarization using linguistic knowledge acquisition," *Information Processing & Management*, vol. 25, no. 4, pp. 419–428, 1989.

[29] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "The detection and classification of non-functional requirements with application to early aspects," in *Proceedings of the 14th IEEE International Requirements Engineering Conference*, ser. RE '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 36–45. [Online]. Available: http://dx.doi.org/10.1109/RE.2006.65

[30] B. Klimt and Y. Yang, "The enron corpus: A new dataset for email classification research," in *European Conference on Machine Learning*. Springer, 2004, pp. 217–226.

[31] C.-W. Hsu, C.-C. Chang, C.-J. Lin *et al.*, "A practical guide to support vector classification," 2003.

[32] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM Sigkdd Explorations Newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[33] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[34] M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and their Applications*, vol. 13, no. 4, pp. 18–28, 1998.

[35] D. W. Hosmer Jr and S. Lemeshow, *Applied logistic regression*. John Wiley & Sons, 2004.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[37] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: ACM, 2006, pp. 161–168. [Online]. Available: http://doi.acm.org/10.1145/1143844.1143865

[38] G. Carenini, R. T. Ng, and X. Zhou, "Summarizing emails with conversational cohesion and subjectivity." 2008.

[39] C. J. V. Rijsbergen, *Information Retrieval*, 2nd ed. Newton, MA, USA: Butterworth-Heinemann, 1979.