

See discussions, stats, and author profiles for this publication at: <http://www.researchgate.net/publication/230754035>

Detecting and Classifying Patterns of Requirements Clarifications

CONFERENCE PAPER · SEPTEMBER 2012

DOI: 10.1109/RE.2012.6345811

CITATIONS

7

READS

78

4 AUTHORS, INCLUDING:



Eric Knauss

University of Gothenburg

70 PUBLICATIONS **272** CITATIONS

SEE PROFILE



Jane Cleland-Huang

DePaul University

149 PUBLICATIONS **1,934** CITATIONS

SEE PROFILE

Detecting and Classifying Patterns of Requirements Clarifications

Eric Knauss, Daniela Damian, Germán Poo-Caamaño, and Jane Cleland-Huang[†]

SEGAL, University of Victoria, Victoria B.C., Canada

knauss@computer.org, gpoo@calcifer.org, danielad@cs.uvic.ca

[†]DePaul University Chicago

jhuang@cs.depaul.edu

Abstract—In current project environments, requirements often evolve throughout the project and are worked on by stakeholders in large and distributed teams. Such teams often use online tools such as mailing lists, bug tracking systems or online discussion forums to communicate, clarify or coordinate work on requirements. In this kind of environment, the expected evolution from initial idea, through clarification, to a stable requirement, often stagnates. When project managers are not aware of underlying problems, development may proceed before requirements are fully understood and stabilized, leading to numerous implementation issues and often resulting in the need for early redesign and modification.

In this paper, we present an approach to analyzing online requirements communication and a method for the detection and classification of clarification events in requirement discussions. We used our approach to analyze online requirements communication in the IBM[®] Rational Team Concert[®] (RTC) project and identified a set of six clarification patterns. Since a predominant amount of clarifications through the lifetime of a requirement is often indicative of problematic requirements, our approach lends support to project managers to assess, in real-time, the state of discussions around a requirement and promptly react to requirements problems.

Keywords—requirements clarification patterns; distributed requirements engineering; communication of requirements

I. INTRODUCTION

In large software projects stakeholders often need to collaborate across geographically distributed sites and to rely upon online communication to perform requirements related activities. Agile software teams in particular implement iterative processes to requirements discovery and use frequent communication instead of requirements documentation. Requirements are defined in the form of user stories, and ongoing discussions around user stories serve as the main mechanism to clarify the meaning of requirements and to coordinate their implementation [1]. Whether the time zone differences between stakeholders are too great to allow for frequent synchronous interaction, or the project mandates the recording of project communication online [2], online project repositories contain a wealth of requirements-related communication, making them valuable for research on communication in requirements engineering. IBM[®]'s Rational Team Concert[®] project, with a large distributed team, is an example in which management mandates the recording of all decisions in the project repository for future

use in the project [2].

In these kinds of environments, however, the expected evolution of a requirement from an initial idea, through clarification, to design and full implementation, often stagnates. Often stakeholders continue to *clarify* the requirement because it is ambiguous, incomplete, or has frequent changes. As a result, its implementation can be delayed or sometimes never get started. Bikeshedding, also known as the Parkinson's law of triviality [3] is another common situation in which developers give disproportionate weight and time to solving trivial issues and delay development. An example from the RTC project (jazz.net) is the ongoing discussion of a large number of developers over the text required in a UI element, and which blocked the development of this user story. Late in the project a manager intervenes and makes a decision "we go with [...] for this iteration", after which development of the story is completed. Although with a happy ending, many situations like this go unnoticed by managers. Current requirements management tools offer little support for identifying requirements with progression problems, thereby lowering the project manager's ability to intervene in a timely manner.

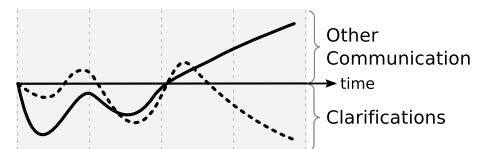


Figure 1: Two different trajectories of reqts. communication

Studying recorded online communication fills this gap by offering the potential to reveal patterns of communication that correlate to problematic situations around requirements development. In this paper we describe a novel approach for differentiating between healthy and problematic patterns of communication associated with an individual requirement. We analyze the content of communication among stakeholders involved in the discussion of a particular requirement, identify specific instances of *clarifying* communication, and examine the trajectory of clarifications (i.e. amount and progression) throughout the lifetime of a requirement. Figure 1 shows two distinct and quite different possible trajectories

of *clarifying* communication in the lifetime of a requirement. While one would expect clarification to diminish as development of a requirement nears the end (solid line trajectory), its predominance throughout the requirement's life may be indicative of problematic requirements (dashed line trajectory). The method proposed in this paper aims to identify these patterns automatically so that managers or involved stakeholders can be made aware of requirements that should be closely investigated.

The contribution of this paper is the method for the detection and classification of clarification communication patterns, as well as a set of six communication patterns that we identified by applying our method in a large industrial project. The remainder of the paper is structured as follows: Section II surveys related work in the study of communication in requirements engineering (RE), as well as related to information retrieval and automated classification techniques in RE. Section III introduces our research approach in studying requirements communication and the set of our techniques for the detection and classification of clarification patterns. We then describe the details of the industrial case study in which we applied our techniques in Sections IV, V, VI and VII, and conclude with future research steps in Section VIII.

II. RELATED WORK

There is a growing body of research that studies communication in requirements engineering. While most papers identify problems in RE communication [4] or model collaboration between stakeholders in requirements-centric teams (e.g. [5]), only a few investigate the actual instances and *content* of communication between stakeholders. Studies by Damian and colleagues [5] found that clarification of requirements and communication of changes are the most predominant topics of discussions. Calefato et al. [6] examined the content of requirements communication to compare the level of common ground achieved in computer-supported elicitations vs. negotiations and used certain heuristics to identify clarification communication. Another recent study in agile teams [7] observed verbal communication about requirements in co-located agile teams. The study revealed a number of communication practices related to memory, communication, and learning in a process of shared conceptualization and which supported the team in its RE activities. In this paper we analyze the content of requirements-related communication as available in online project repositories.

Our work also closely relates to research into automatic classification of textual descriptions of requirements. Cleland-Huang developed a technique for detecting various types of non-functional requirements, such as security, performance, and usability requirements, from large requirements specifications [8], [9]. Their approach trained a classifier to recognize a set of weighted indicator terms, indicative of each type of requirement. Kof, Lee et al. worked on

extracting semantics from natural language texts [10], [11] to identify ambiguities in requirements specifications and focused on the semi automatic extraction of an ontology from a requirements document. Chantree et al. described how to detect nocuous ambiguities in natural language requirements [12] by using word distribution in requirements to train heuristic classifiers (i.e. how to interpret the conjunctions *and/or* in natural language). The process of creating the dataset is very similar to our work, i.e. the collection and classification of realistic samples based on the judgement of multiple experts to enhance the quality of the dataset.

III. RESEARCH APPROACH

Requirements elicitation, analysis and negotiation engage stakeholders in a process of continuous gathering, clarification and interpretation of requirements. A human-centred process, RE is a process of communication among stakeholders [13] in a quest for shared understanding of requirements [14]. While at the initial formulation of a requirement, its discussion focuses on *clarifying* the requirement, e.g. removing ambiguities or gathering more information about the stakeholders and their goals, while communication in the later stages of the project may relate to coordination actions as developers implement the requirements and supposedly have a greater understanding of the requirement. One would expect, then, that in a successful implementation of a requirement, the amount of requirement clarification would diminish as its implementation nears the end. A requirement for which stakeholders continue to have significant clarification communication in later stages of a project indicates that the requirement development has problems, because it either stagnates or is entirely not feasible.

Our approach is to analyze the discussion thread for individual requirements and to identify discussion events in which stakeholders are seeking *clarification* on the requirement as opposed to other communication that relates to downstream activities (e.g. design, implementation, testing) or their coordinative actions. Our long term goal is to develop a tool-supported approach that analyzes online documented discussions in order to help managers pinpoint requirements that are failing to progress. In this paper we describe our first stage in this research, which includes the development of a clarification classifier and an initial validation in an industrial project. Table I defines the terms that we use to describe our two research questions:

- RQ1: Can we identify patterns of *clarification* in *requirements discussions*?
- RQ2: Can we detect and classify these *clarification patterns* automatically?

To answer these questions we build and conduct an initial validation of a clarification classifier. Our proposed method, which is depicted in Figure 2 and described fully in the sections below, includes the four primary phases of manual *classification of requirements discussion events, building an*

Table I: Definitions in our Approach

<i>Requirements discussion:</i>	The thread of communication that is related to a given requirement. This includes communication found in requirements activities such as elicitation, negotiation or validation of a requirement, or in downstream activities related to the particular requirement, such as its design, implementation or testing. It consists of a series of <i>discussion events</i> , i.e. contributions to the discussion.
<i>Clarifications:</i>	A discussion event in which the discussant seeks to improve the understanding of the requirement by either asking for clarification or by offering additional information that make the requirement clearer.
<i>Clarification pattern:</i>	A reoccurring trajectory of communication throughout the lifetime of a requirement in which we purposefully distinguish clarifications from non-clarification events. Figure 1 shows two possible trajectories that we identify in this paper: the <i>textbook-example</i> pattern (solid line) and the <i>back-to-draft</i> pattern (dashed line).

automatic clarification classifier for these events, *development of clarification patterns*, and *automatic pattern identification*. During the first phase we investigate, whether human raters can identify clarification in requirements discussions (see Section IV). In this iterative process, a classification scheme is gradually refined until a stable set of classified discussion events is created. This set is then used to evaluate whether automatic classifiers can be trained to identify clarification in requirements discussions, as discussed in Section IV-A. It is also used to construct visualizations and to create a clarification pattern catalogue (Section V). Finally, we explore how to use automatically classified discussion events and the clarification pattern catalogue to identify clarification patterns in software projects (Section VI).

As an initial validation of our method we applied it in an industrial case study with a large repository of project communication that could be traced to requirements. We describe background information on the project in the next section. We then describe in detail each of the method phases in the context of their application in the industrial project.

A. IBM's Rational Team Concert project

IBM Rational Team Concert (RTC) is a collaborative software development environment built on the JazzTM architecture. It integrates its own source-code control system, an issue-tracking system, and collaborative development tools.

At the time of our study, the RTC development team involved 151 active developers distributed over 16 different sites located in the United States, Canada, and Europe. The project uses the agile and iterative Eclipse Way development process [2] with six-week iteration cycles. A project management committee formulates the goals and features for each release at the beginning of the each iteration, and stores them as *stories* in the system.

One unique aspect of RTC is that it is self-hosted (the

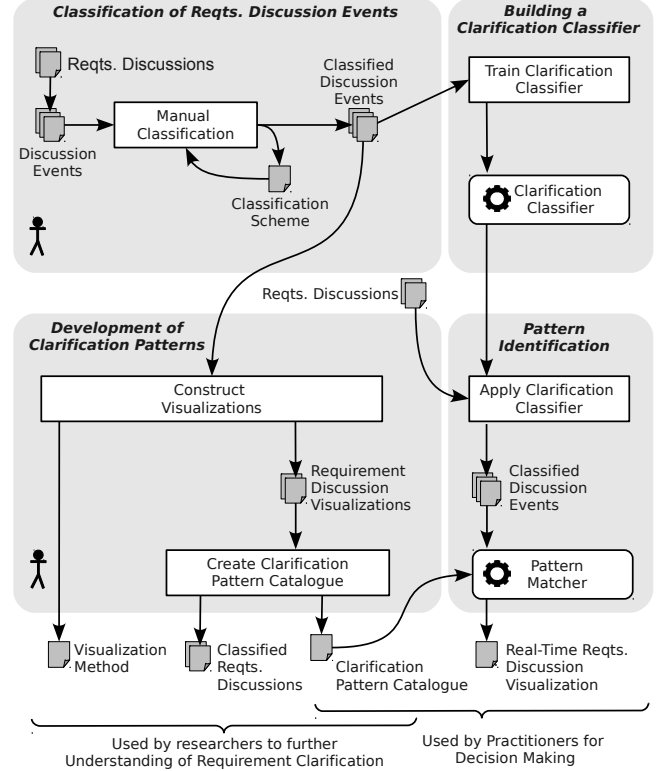


Figure 2: Overview of the approach to analyze *Patterns of Requirements Clarification*.

team uses its own software to develop subsequent versions) and follows an "open-commercial" development model. Its features are proposed both by its own developers as well as outside customers. This is done through its publicly-accessible issue-tracking system, through which outside customers have access to the features (stories) and actively participate in their discussion. Given the distributed nature of the team's interaction, communication between individuals is predominantly online through *comments* in the issue-tracking system, though they also use instant messaging, Internet Relay Chat, phone, and face-to-face. Personal email is discouraged. A best practice in the RTC project is that developers explicitly record communication at local sites as *comments* in the system so that it is available to the entire distributed team.

We chose RTC for the application of our method because it has a rich issue-tracking repository in which communication about its requirements is documented online (in the form of comments) and is traceable to requirements.

In our case study we analyzed a snapshot of the issue tracking system between December 2006 and June 2008. The snapshot contains 157 stories. The conceptualization of the terms in our method in the RTC study is as follows: since these *stories* are *requirements* in RTC, the *comments* associated with each story represent the *discussion events*

Table II: Statistics of the 97 RTC requirements discussions.

	<i>Avg</i>	<i>Min</i>	<i>Max</i>
Time length per reqts. discussion [days]	63.4	1	479
Comments per reqts. discussion	11.8	1	139
Contributors per reqts. discussion	3.7	1	17

about the story, and the thread of comments associated with the story represents its *requirements discussion*.

Out of the 157 stories, 60 did not have any comments, and 36 were complex stories had a number of associated sub-issues (e.g. stories or tasks). Following an agile process, the RTC team often continues the discussion of stories in their associated sub-issues and thus we included comments from sub-issues into the respective story’s discussion so that we account for the story’s complete set of discussion events. In total we thus analyzed 97 stories and their requirements discussions with a total of 1159 comments (see statistics of the requirement discussions for the 97 stories in RTC in Table II).

IV. CLASSIFICATION OF REQUIREMENTS DISCUSSION EVENTS

The first and crucial step in detecting and classifying patterns of clarification is to identify requirements discussion events that deal with clarification. Thus, in this first stage of our method a *classification scheme* that distinguishes between *clarification* and *other* discussion events is developed. The classification scheme is developed manually and through an iterative process of reading discussion events in the requirements discussions and classifying those events as *clarification* or not. Table III shows the classification scheme developed in the RTC project, in which we created subcategories intended to help the human analyst more easily categorize the discussion event as clarification or not. The *clarification* subcategories that emerged were related to typical RE activities [13] such as requirements elicitations, requirements interpretation, negotiation, documentation, verification, and validation. The *Other* communication subcategories related to *coordination* activities in the implementation of the requirement (the four coordination types as identified in [15] as well as *coordination response*), in the design and implementation of the requirements (*solution engineering*), general *announcements* or *off topic* comments. For each of these subcategories we include example comments from the RTC dataset in Table III.

This classification scheme drives the manual classification of discussion events. In RTC, two raters worked together to classify a total of 1159 comments and to develop a stable classification scheme. As some comments, for example, mentioning a new requirement, could be considered either as clarification or as coordination II, the raters developed a set of heuristics to guide the classification process. For example,

in this case, a heuristic was created that only clarification of requirements should be taken into account.

The iterative process of classifying comments in RTC was as follows: First, the two raters classified comments together to create an initial classification scheme. Then each rater classified a specified set of comments individually. The agreement in rating a given comment as *clarification* or not in this set of comments was computed based on Cohen’s Kappa (κ) [16]. If the agreement was too low ($\kappa < 0.7$), the comments where the raters disagreed were discussed, the classification scheme was improved, and the raters started again with classifying comments.

The two raters reached a satisfactory result after three iterations. They agreed on the classification of 378 comments from a total of 449 comments classified by both raters, thus reaching a satisfactory kappa ($\kappa \approx 0.7$). Based on this result, each of the remaining 710 comments was only classified by one of the raters.

A. Automatic Classification

Our method for automatic classification of requirements discussion uses one of the most basic machine learning algorithms: the Naive Bayesian classifier. More specifically, we use an implementation based on the popular descriptions of such a classifier [17], and which we successfully used in our previous research to identify security-relevant requirements in software requirements specifications [18].

In RTC, the set of classified comments was used to train and evaluate the automated classifier. We use a standard 10-fold cross validation technique [19] in which the discussion events in the dataset are divided into 10 equal sets. Nine sets are then used to train the classifier, which is then used to classify the results of the tenth set. This is repeated ten times using each possible combination, until each of the ten sets has been tested one time. We then compute standard metrics from information retrieval [20] (i.e. recall, precision, specificity, and f-measure) for each of the ten runs and derive the average performance of the automatic classifier.

In the RTC project the results of the 10-fold cross validation are shown in Table IV: a recall of 0.943, a precision of 0.678, a f-measure of 0.789, and a specificity of 0.552. A reference classifier that would simply classify every comment as clarification would have an recall of 1, a precision of 0.5, and a f-measure of 0.667.

In Section VI-A we specifically address the question of whether these results are sufficiently accurate to identify clarification patterns.

V. DEVELOPMENT OF CLARIFICATION PATTERNS

With the ability to classify requirements discussion events, the third phase in our method is to develop clarification patterns of requirements discussions.

We first use techniques to visualize the classified discussion events over time for each requirement discussion. Then

Table III: Classification Scheme

Category	Subcategory	Example comment in the RTC dataset
Clarification	<i>Req. Elicitation</i> : Add (or ask for) a requirement or a stakeholder.	"If I am not mistaken, we have two server machines. Why are two additional machines needed? I would have expected only one additional machine would have been needed." "A large oil company on the UK is very interested in this as well - as they have a heterogeneous environment using both [...]"
	<i>Req. Interpretation</i> : Derive conclusions or sub requirements from a given requirement	"Some configuration details that may have been assumed that I would like to make specific, explicitly suggestions for: [...]"
	<i>Req. Negotiation</i> : discuss alternatives and priorities of requirements	"I do care. When I have more than 1 conflict, I do [...] I would be interested in the hints on the file in an outgoing changeset. I would likely then just open the compare editor and verify my changes."
	<i>Req. Documentation</i> : give a report of the requirements discussed in this thread.	"Based on discussions, the tactical requirement here is just to add this slot for PermissionDeniedException."
	<i>Req. Verification/Validation</i> : discuss whether a given requirement was understood correctly.	"Wasn't the point of this to allow the caller to decide if the exception should even show in the TA? [...] My understanding was that the TA was to show process related expected errors, such as [...]"
Other	<i>Give context</i> : add a screenshot or longer description to give context for the requirements.	"Defect <defect id> may report the same underlying issue, with a stack trace but without the screen shot."
	<i>Coordination I</i> : Managing shared resources: Instructing or suggesting a person to perform a task.	"<name 2>, can you attach in a screenshot of the 'ClearCase Synchronized Streams' view?"
	<i>Coordination II</i> : Managing producer/consumer relationships: The creation or dissemination of information (Remark: includes announcing that a story is completed).	"Note: We are reviewing this proposed change tomorrow at the 1pm EDT CC Connector team meeting. If anyone would like to dial in, please let me know and I'll set up a call."
	<i>Coordination III</i> : Managing simultaneity constraints: Synchronizing tasks between actors. Taking possible times for an event(s). Allocating a time for a particular event(s). Passing information about the time of an event(s).	"Since you're not going to get to this for <milestone 1>, I'll take it back for <milestone 2>."
	<i>Coordination IV</i> : Managing task / subtask dependencies: Planning tasks and strategy to achieve the overall goal.	"Work item <defect id> was extracted from this work item."
	<i>Coordination response</i> : acknowledge a coordination item, e.g. that a task has been performed.	"Yes, I have planned time for this, sorry for the postponing..."
	<i>Solution engineering</i> : contribute directly to the implementation of this requirement	"I have delivered changes for this to work on Windows. Need to make it work on UNIX next."
	<i>Announcement</i> : Announce something that is not directly related to coordination of this workitem	"A first draft of the wiki page is available: <url>"
	<i>Offtopic</i> : Something unrelated to this workitem or the project in general.	"Adapting to new story work flow."

Table IV: Results of automatic classification

	Clarification	Other
from manual classification	579 (<i>relevant</i>)	580
automatically classified as clarification	54.6 (<i>true positive</i>)	26.0 (<i>false positive</i>)
automatically classified as other	26.0 (<i>false negative</i>)	32.0 (<i>true negative</i>)
$\text{precision} = \frac{ \text{true positive} }{ \text{true positive} + \text{false positive} } = 0.678$ $\text{recall} = \frac{ \text{true positive} }{ \text{true positive} + \text{false negative} } = 0.943$ $\text{specificity} = \frac{ \text{true negative} }{ \text{true negative} + \text{false positive} } = 0.552$ $\text{f-measure} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = 0.789$		

we aggregate these visualizations in distinguishable classes that we call patterns.

A. Visualizing Clarification in Requirements Discussions

A meaningful visualization is the prerequisite for the identification of patterns. We built a visualization tool and illustrate the visualization of the classified comments in the bikeshedding scenario in the RTC project in Figure 3. The horizontal *timeline* represents time from the creation

of a requirement (story in RTC) to the current date (its final implementation in this case), 189 days. The 121 classified discussion events (comments in RTC) from 14 different contributors are then represented below or above this timeline: the *clarification* comments are shown as lighter coloured boxes below the timeline; comments classified as *other* communication are represented as black boxes and stacked above the timeline. This was a complex story in our dataset and we visualize the comments for all its 8 associated sub issues (in which the user story discussion continued) on the same timeline. Because there could be many discussion events in a requirement timeline, we choose to define equal partitions of the horizontal axis (in this case 32 partitions), and stack the discussion event boxes on top of each other if they occur in the same partition. The choice of the number of partitions and the associated aggregation of discussion events within partitions is an important and challenging research problem that we discuss later in the Discussion section.

B. Deriving Requirements Clarification Patterns

A *clarification pattern* is an abstract, reoccurring trajectory of communication throughout the lifetime of a

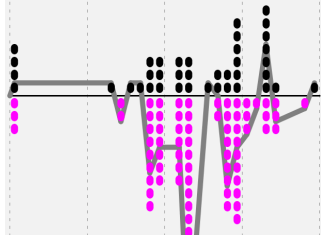


Figure 3: Trajectory of clarification and other comments in the bike shedding scenario

requirement. In the RTC project, we initially visualized the trajectories of communication for simple stories in our data set, i.e. the stories that did not have associated sub issues, 61 in total. We manually inspected all visualizations and searched for reoccurring patterns. Then we constructed the visualizations for the 36 remaining complex stories (those that had associated sub issues) and found that these complex stories fit into our existing patterns, with only three exceptions.

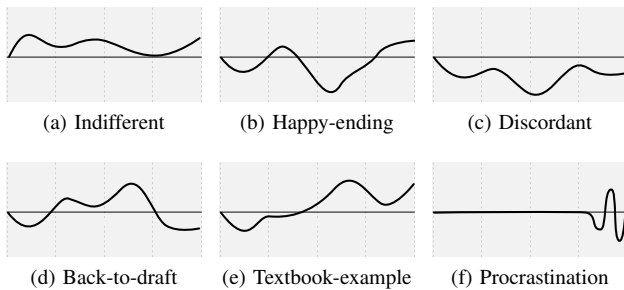


Figure 4: Clarification patterns identified in the RTC project.

Figure 4 shows six clarification patterns that we describe below (number of instances found in RTC are shown in brackets). The bikeshedding example falls into the *happy-ending* pattern.

1) *Indifferent* (36 instances): Requirements in this pattern show no evidence of clarifications on the requirement. Our *heuristic* for putting them in this pattern was that they have no associated clarification events. Figure 5a shows two representatives of this pattern. Depending on the trust the project manager has in her team, requirements in this class do not need any more attention, because there seems to be no need to clarify the requirements. However, it could be that the developers dealing with this story are reluctant to openly discuss their questions or to approach the end-users.

2) *Happy-ending* (16 instances): This pattern contains stories that might fit into more suspicious classes (e.g. *discordant* and *back-to-draft*), if it was not for the last few comments. Figure 5c gives typical examples, where the level of shared understanding returns to being positive

in the very end. Our *heuristic* for this class is that the amount of clarification was greater or equal to the amount of other discussion events in at least one of the partitions in the second half and that the last non empty partition has less clarification than other discussion events. It is possible that the responsible subteam recovered from a *back-to-draft* situation. Depending on the situation it is also possible that the team is still in the clarification phase and that the requirements might reach a stable state in the near future. Despite the name, it is also possible that the team is doing some desperate coordination to get this story in the next milestone despite vague and contradicting requirements.

3) *Discordant* (14 instances): Requirements in this class show a predominance of clarification during their entire lifetime, and our *heuristic* is that in every partition the number of clarifications is higher or equal to the number of other discussion events. Project managers might accept this, if the project or iteration is just starting. However, a requirement that never leaves this state should be closely investigated. Figure 5e shows two examples from the 14 instances of this pattern in our data.

4) *Back-to-draft* (12 instances): Requirements in this class originally looked good when the team started to implement them. However, during the work, the discussion snaps back to clarification. The examples in Figure 5g illustrate this pattern. Our *heuristic* for this class is that during the trajectory of the discussion there is at least one partition with less clarification than other discussion events and that the last non-empty partition has a greater or equal amount of clarification. Depending on how much time is left to complete this story, project managers might want to investigate if this is a problem for the schedule.

5) *Textbook-example* (11 instances): As shown in Figure 5i, some stories show a classic trajectory of clarification from mostly clarification in the first half of the timeline to mostly downstream activities in the second half. Our *heuristic* for putting stories into this class are: there is clarification in the first half of the timeline, there is no partition with more clarification than other discussion events in the second half, and in total there is less clarification than other communication in the second half.

6) *Procrastination* (5 instances): For this pattern, the discussion of stories only starts at the end of the timeline. Because for this pattern most of the comments fall into the same partition deriving meaningful patterns based on our standard visualization is challenging. Nevertheless, these stories look suspicious and project managers might want to have a second look at such cases. Figure 5k illustrates two out of five examples we found for this pattern. Our *heuristic* for this class is that all comments are located in the last quarter of the timeline and the visualization does not indicate a *discordant* or *indifferent* pattern.

LEFT: IDENTIFIED PATTERNS

RIGHT: RESULT OF AUTOMATIC PATTERN IDENTIFICATION

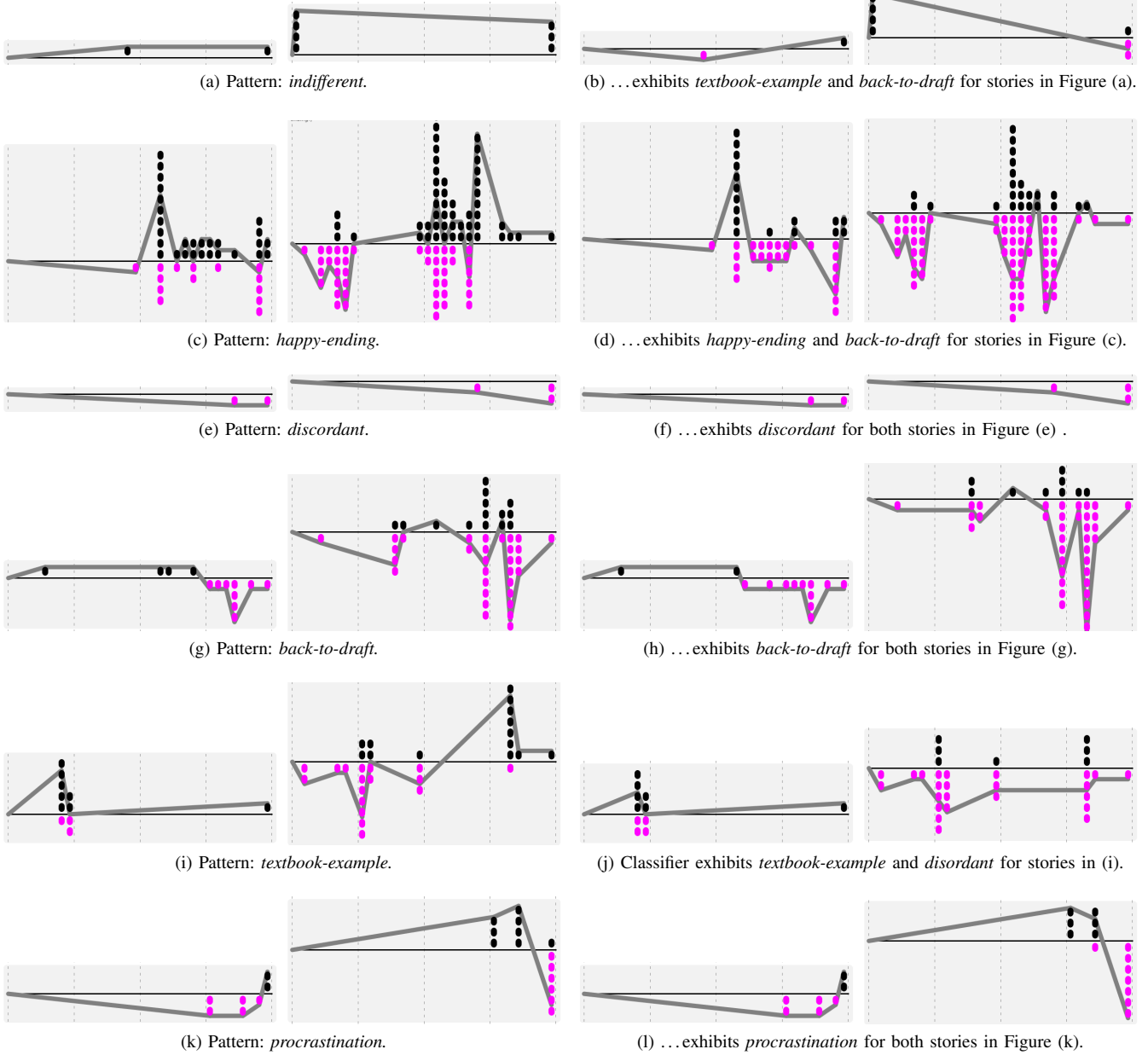


Figure 5: Six clarification patterns (in rows) identified in the RTC project. Two instances from manual classification (left) and automatic classification (right) are shown for each pattern.

VI. AUTOMATIC IDENTIFICATION OF CLARIFICATION PATTERNS

In the fourth phase of our method we create an automatic *pattern matcher* that implements the heuristics defined in the previous phase and identifies the pattern for automatically classified requirements discussion. In this section, we give a preliminary evaluation of the performance of the clarification classifier in combination with the pattern matcher. From the 10-fold cross validation we know that the clarification

classifier adds a clarification item to the visualization in 94% of the cases the raters did (because of the high recall). The classifier adds a clarification item to the visualization in about 32% of the cases where the raters added a non-clarification item (because of the low precision). We now investigate the impact these properties have on our ability to automatically identify clarification patterns.

A. Preliminary Evaluation Method

We applied the clarification classifier and pattern matcher on RTC stories and measured how often the identified pattern matches the pattern exhibited by the manually constructed visualizations on a selected sample of requirements discussions. The sample included three examples for each pattern: those stories with the maximum, minimum, and average number of comments in each pattern class (only the average and maximum example is shown in Figure 5). Note, that these numbers differ for each pattern. We then trained the clarification classifier with the comments in our dataset that did not belong to any story in the sample. We excluded the two stories with more than 100 comments, to avoid the possibility that our training set might be too small to be representative. Then we automatically classified the examples in the sample, let the pattern matcher identify the patterns and compared the results with the patterns exhibited by the visualizations from the manual classifications.

B. Preliminary Evaluation Results

Table V gives the results of our preliminary evaluation in a confusion matrix. Each column represents one of our patterns. Each row reports the results from applying the clarification classifier and the pattern matcher. The numbers in the main diagonal (bold) represent the true positives identified by the classifier. The other numbers show the false positives (rows) and false negatives (columns) for each pattern. The last two columns give recall and precision for automatically identifying the pattern.

1) *Indifferent*: This is the most fragile pattern with respect to our classifier, because it is defined as having no clarification. The more events the story has, the more likely it is that one gets mistaken to be clarification, as seen in Figure 5b which looks now like a *textbook-example* on the left and *back-to-draft* on the right (recall = 0).

2) *Happy-ending*: Automatic pattern identification works well for the minimum and average sized example, but, as Figure 5d shows, our automatic classifier mistakenly detected a rather bad ending for the more complex example on the right side, qualifying this story as a *back-to-draft* story. This mistake might not be too bad, because the *happy-ending* pattern is only slightly less suspicious than the *back-to-draft* pattern. Overall, 2 out of three happy-ending stories were correctly identified and no non happy-ending story was mistaken for a happy-ending (recall = 0.667, precision = 1).

3) *Discordant*: Based on the high recall, all clarification events are correctly identified, thus the pattern is correctly identified in all three stories (recall = 1). 4 out of 15 non discordant stories were classified mistakenly as discordant, leading to a precision of 0.429.

4) *Procrastination*: As this class is defined rather by the time of events then by the type of events, this class is robust against the faults of our clarification classifier. The only likely fault would occur if all events are classified as 'other',

in which case, the story would be classified as indifferent. Because of the high recall, this is very unlikely and it is impossible to mistakenly put a non procrastination story in this class (recall = 1, precision = 1).

5) *Back-to-draft*: For the same reason, the *back-to-draft* class also has a high recall. Figure 5h shows how stories still fit into this class, even though some events were falsely classified as clarification. This behaviour causes 2 out of 15 non back-to-draft stories to be identified as back-to-draft, leading to a precision = 0.5; however 2 out of 3 back-to-draft stories were correctly identified (recall = 0.667).

6) *Textbook-example*: The average sized example works well, but the minimum and maximum examples are not successfully identified. In the complex example, too many false positives make the requirements discussion look like a *discordant* story (see Figure 5j, right side). 1 out of 3 textbook-examples were correctly identified (recall = 0.333), and 1 out of 15 non textbook-examples was mistakenly identified (precision = 0.5).

Table V: Patterns: number of instances and validation results.

Pattern	Indifferent	Happy-ending	Discordant	Back-to-draft	Textbook-exam	Procrastination	none	Precision	Recall
Indifferent								NaN	0.000
Happy-ending		2						1.000	0.667
Discordant	2		3	2				0.429	1.000
Back-to-draft	1	1		2				0.500	0.667
Textbook-example			1	1				0.500	0.333
Procrastination					3			1.000	1.000
none									

VII. DISCUSSION

This paper introduced a new approach to investigate requirements related communication aimed at supporting managers and other stakeholders identify requirements that may need attention because they exhibit problematic patterns of communication. Our approach to this problem was to develop a method to detect and classify *clarification* patterns in requirements discussions. By identifying requirements for which communication is predominantly about clarifying the requirement instead of design or implementation activities, managers are given support tools to spot problematic requirements. Our approach was driven by two research questions. In the first question we investigated whether patterns of clarification can be identified in requirements discussions. We developed a method that combines manual and automated classification of *clarifications* in requirements discussions. We applied it to the analysis of requirements discussions in IBM's RTC project and identified six clarification patterns. Our second research question tackled the automated classification of these clarification patterns. We used our automatic classifier to automatically detect these

patterns in the RTC communication data set with positive initial validation results. In this section we discuss the contribution of this work, its implications for research and practice, as well as threats in the validation of our classifier in the RTC project.

A. Implications for research

The first step in the method presented in this paper makes use of a visualization method to examine and classify requirements discussions. This visualization method and the classified requirements discussions in the RTC project can be used by RE researchers to further our understanding of requirements communication and specifically *clarifications* in requirements discussions. For example, requirements discussions that are classified using our method can be used to formulate or test research hypotheses about requirements communication and its relationship to other project factors. At the same time, the development of such visualizations of requirements discussions poses challenging research questions in itself. The construction of visualizations has significant impact on the type of patterns that can be derived. The decision on the number of partitions on the timeline and the subsequent aggregation of discussion events over time, can change the predominance of clarification in a given section of the timeline. In the RTC project, we experimented with dividing the timeline into 8, 32, 64, and unlimited partitions, with the effect that three stories show different patterns with 8 partitions. However, suitable aggregation of data should be investigated through usability studies with software practitioners. Validating the visualization effectiveness with RTC managers is a next step in our research.

The clarification patterns identified in the RTC project are another contribution of this work. Our pattern classification method uses a data-driven approach to developing the patterns of clarifications in a project. In the RTC project we did not have any preconceived idea of how these patterns would look except that they would exhibit a combination of clarification and other communication, and that some would indicate predominant clarification communication throughout the project. The six patterns we identified are quite varied and range from a complete lack of clarification events as in the *indifferent* pattern, to the *textbook-example* pattern and the *procrastination* pattern. While a bit more than a third of our data set fits into the *indifferent* pattern, the other patterns have relatively equal number of instances, with the exception of the *procrastination* pattern that has fewest instances. It is not surprising that we did not find a higher number of instances in the *textbook-example* as one would want. Following an agile development process, the RTC practice is characterized by a continuous discussion of the requirements and that is reflected in a comparable number of instances of the *happy-ending* pattern.

However, our method did identify approximately the same number of instances in two other patterns that could possibly

be problematic, the *discordant* and *back-to-draft* patterns.

While these patterns provide insights into the RTC team, future research should attempt to use our method to replicate these patterns in other large distributed projects that record their requirements discussions, and which use similar as well as other development methodologies. Open source projects for example also maintain a wealth of project and requirements discussion data in their mailing lists.

B. Implications for practitioners

The practical goal of our method is to support managers in identifying requirements that exhibit problematic patterns of communication. Our method is not only intended for real-time application in a project to help diagnose requirements under development, but also to enhance project specific information. First, software practitioners can apply this method to analyze historical communication records and identify a catalogue of patterns in their project or organization. This may reveal particular communication practices that are process specific, or identify unexpected work practices. This information supports managers in decision-making with respect to supporting tools or process methodologies. Second, the application in real-time of our automatic pattern classifier to analyze *current ongoing* requirements discussions supports managers in examining the health of a requirement development based on the trajectory of clarification relative to other communication about the requirement. Project specific information such as development process, communication practices and tools allow the manager to decide whether the respective requirement does need attention and thus make timely decisions. The case shown in Figure 3 is a clear example of a situation where a manager benefits from details concerning the communication about a requirement and makes a decision in real time to end the clarification discussion and proceed with development of the respective requirement. Unfortunately managers cannot participate in all requirements discussions in a project and our method provides an automated solution for identifying requirements with such problematic development.

C. Threats to Validity

1) *Construct Validity*: One threat in our study is that the contributors to the RTC discussions do not discuss requirements online. Our two raters read a significant amount of comments in our data set and we are confident that the RTC issues contain requirements discussions and therefore are a valid object of investigation. However, some of the comments were more complex than others and future work should investigate whether the mapping of comments to discussion events is valid or if complex comments should be broken down into simple communication events.

2) *External and Conclusion Validity*: Our evaluation is exclusively based on data from the RTC team and our knowledge about the applicability of this approach to other projects

is therefore limited. Open source projects also have online requirements communications recorded in email messages. As more and more requirements engineering approaches try to combine different sources of requirements (e.g. chat, email, bug-databases), we assume that more projects will become suitable for this kind of analysis, especially if they are developed in large and distributed teams that have to rely on online communication. Furthermore, we expect that our clarification patterns are project and process dependent, e.g. the high number of *procrastination* discussions could be the result of RTC's agile process, which causes the discussion of low-priority items to start only late, though we believe our findings are applicable in any iterative development project.

VIII. CONCLUSION

The main contribution of this paper is the identification of six different clarification communication patterns identified in online communication related to requirements, and a novel technique for automatically classifying user stories into the six pattern categories. As an increasing number of projects are globally distributed and therefore forced to rely upon online communication mechanisms, our approach can lead to a more effective and productive development environment in which project managers are given up to date information on potentially problematic requirements.

In our future work we intend to conduct evaluation of our classifier in a different domain and project environment. We also plan to conduct a field evaluation with RTC managers to evaluate the usefulness of our approach in a real project, and to investigate how results from our classifier can best be presented to decision makers.

IX. ACKNOWLEDGMENTS

We thank the RTC team members who provided clarifications on our dataset when needed, and the SEGAL Lab members for their feedback on many versions of this paper.

REFERENCES

- [1] L. Cao and B. Ramesh, "Agile requirements engineering practices: An empirical study," *Software, IEEE*, vol. 25, no. 1, pp. 60–67, jan.-feb. 2008.
- [2] R. Frost, "Jazz and the eclipse way of collaboration," *IEEE Software*, vol. 24, no. 06, pp. 114–117, 2007.
- [3] C. N. Parkinson, *Parkinson's Law: The Pursuit of Progress*. John Murray, 1958.
- [4] A. Al-Rawas and S. M. Easterbrook, "A field study into the communications problems in requirements engineering," in *Proc. of Conf. on Professional Awareness in Software Engineering*, London, 1996.
- [5] D. Damian, I. Kwan, and S. Marczak, "Requirements-driven collaboration: Leveraging the invisible relationships between requirements and people," in *Collaborative Software Engineering*, I. Mistrík, A. van der Hoek, J. Grundy, and J. Whitehead, Eds. Springer Berlin Heidelberg, 2010, pp. 57–76.
- [6] F. Calefato, D. Damian, and F. Lanubile, "Computer-mediated communication to support distributed requirements elicitation and negotiations tasks," *Empirical Software Engineering*, Oct. 2011.
- [7] N. N. B. Abdullah, S. Honiden, H. Sharp, B. Nuseibeh, and D. Notkin, "Communication patterns of agile requirements engineering," in *Proc. of the 1st Workshop on Agile Requirements Engineering*. New York, USA: ACM, 2011, pp. 1:1–1:4.
- [8] J. Cleland-Huang, R. Settini, and P. Solc, "The Detection and Classification of Non-Functional Requirements with Application to Early Aspects," in *14th IEEE Int'l Requirements Engineering Conf.* IEEE, Sep. 2006, pp. 39–48.
- [9] J. Cleland-Huang, R. Settini, X. Zou, and P. Solc, "Automated classification of non-functional requirements," *Requirements Engineering*, vol. 12, no. 2, pp. 103–120, Mar. 2007.
- [10] L. Kof, "Text Analysis for Requirements Engineering," Ph.D. dissertation, Technische Universität München, München, 2005.
- [11] S. W. Lee, D. Muthurajan, R. A. Gandhi, D. S. Yavagal, and G.-J. Ahn, "Building Decision Support Problem Domain Ontology from Natural Language Requirements for Software Assurance," *Int'l Journal of Software Engineering and Knowledge Engineering*, vol. 16, no. 6, pp. 851–884, 2006.
- [12] F. Chantree, B. Nuseibeh, A. de Roeck, and A. Willis, "Identifying Noduous Ambiguities in Natural Language Requirements," in *Proc. of the 14th IEEE Int'l Requirements Engineering Conf.* Minneapolis, USA: IEEE Computer Society, 2006, pp. 56–65.
- [13] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," in *Proc. of the Conf. on The Future of Software Engineering*. New York, USA: ACM, 2000, pp. 35–46.
- [14] J. Aranda, "A theory of shared understanding for software organizations," Ph.D. dissertation, University of Toronto, 2010.
- [15] L. Hossain, A. Wu, and K. K. S. Chung, "Actor centrality correlates to project based coordination," in *Proc. of the 2006 20th anniversary conference on Computer supported cooperative work (CSCW '06)*. New York, USA: ACM Press, 2006, p. 363.
- [16] J.-W. Strijbos, R. L. Martens, F. J. Prins, and W. M. Jochems, "Content analysis: What are they talking about?" *Computers and Education*, vol. 46, no. 1, pp. 29–48, 2006.
- [17] P. Graham, "A Plan for Spam," 2002, web, January 2011, <http://www.paulgraham.com/spam.html>.
- [18] E. Knauss, S. Houmb, K. Schneider, S. Islam, and J. Jürjens, "Supporting Requirements Engineers in Recognising Security Issues," in *Proc. of the 17th Int'l Working Conf. on Requirements Engineering: Foundation for Software Quality*, D. Berry and X. Franch, Eds. Essen, Germany: Springer, 2011.
- [19] S. M. Weiss and C. A. Kulikowski, *Computer systems that learn : classification and prediction methods from statistics, neural nets, machine learning, and expert systems*. San Mateo, USA: M. Kaufmann Publishers, 1991.
- [20] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. ACM Press, Addison Wesley, 1999.