V:issue:lizer
Explore Online Communication and Requirements Clarification over Time

Eric Knauss, Daniela Damian
SEGAL, University of Victoria, Victoria B.C., Canada
knauss@computer.org, danielad@cs.uvic.ca

***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 the prototype of a tool for analyzing online requirements communication and for supporting our method for the detection and classification of clarification events in requirement discussions. We believe that our prototype will be inspire both researchers and practitioners. Our v:issue:lizer tool enables researchers to analyze data from our previous studies, relate this data to their research, and use our tool to investigate requirements clarification in other research projects. Since a predominant amount of clarifications through the lifetime of a requirement is often indicative of problematic requirements, our v:issue:lizer tool supports 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

Update and Shorten

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.

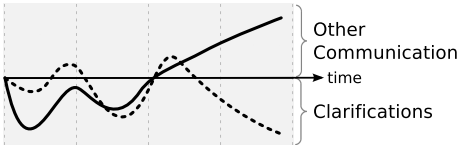

Figure 1: Two different trajectories of reqts. communication

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.

II. Background and Related Work

Harvest RE paper

Read this, change text

Since tasks crosscut both the technical aspects of collaboration an communication, finding the right tasks at the right time is crucial to the success of a project [5]. Treude and Storey identified the lack of visualizations as one of the most important short comings of today's task management systems [6]. They also found that dashboards that report on the state of a task management system can become pivotal to task prioritization in critical project phases.

Read this, change text

Ellis et al. [7] report results from interviews of how developers use Bugzilla, a popular task management system. The motivation for their study was the design of a visualization tool for tasks. One of their key findings was that Bugzilla played a key role in managing the project. Their visualization reveals social and historical patterns in tasks, but it does not focus on exploration activities.

Many related studies focus on mining and analyzing quantitative data to reveal information about the evolution of the system or to predict future behaviours but only few works are concerned with visualizing and exploring this information space. Treude et al. [4] present the workitemexplorer, a highly related tool that allows to explore the information stored in a task management system. Compared to this work, V:Issue:Lizer allows the analysis of a very specialized aspect, i.e. the analysis of online communication related to clarification of requirements.

III. V:Issue:Lizer

Give a screenshot and describe it

V:Issue:Lizer is an interactive tool that allows users to dynamically explore data from a software development task management system with a focus on communication and clarification of requirements related issues. The main control element shows a list of tasks (e.g. workitems in jazz, items in jira, issues in other systems). The power of V:Issue:Lizer is to add visualizations to the selected issues that help to assess the communication in comments related to these issues.

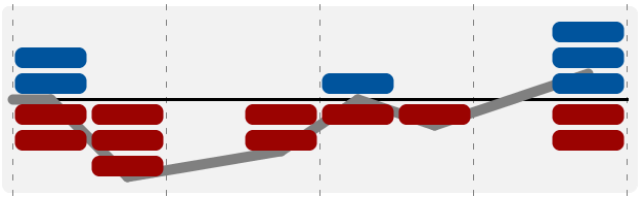

Figure 2: Example of a clarification trajectory

V:Issue:Lizer currently supports two different visualizations:

- **Clarification Trajectories** show how the percentage of communication related to clarifying requirements changes over the lifetime of the issue.
- **Social Networks** show who is participating in a discussion related to (a set of) issue(s) and how the actors in the discussion are structured.

IV. Example Scenarios

In this section we describe examples that highlight the functionality of the V:Issue:Lizer tool.

A. Where are the hotspots in a set of issues?

Often, a clear understanding of requirements only evolves during the development of software. This is especially true (but not limited to) agile software projects, where managers decide to frame only rudimentary requirements and refine the details on the go. For a manager, it is important to know when problematic requirements surface, because they can have a serious impact on the project. V:Issue:Lizer helps managers in this scenario as follows:

1) The manager loads a set of issues (e.g. user stories for the current iteration).
2) V:Issue:Lizer automatically analyzes the comments that are related to these issues and that are available in online communication.
3) V:Issue:Lizer creates a set of *clarification trajectories* (c.f. Figure 2), one for each issue. Comments that are concerned with clarifying requirements are depicted by red rectangles below the timeline, other comments are depicted by blue rectangles above the timeline.
4) V:Issue:Lizer also displays suggestive pattern names for distinctive trajectories (e.g. textbook-example, back-to-draft, procrastination).
5) The manager scrolls through the issues and associated trajectories and decides based on this rich information where to invest more resources.

Typically, there is a number of issues without pathological findings, e.g. user stories with some clarification in the beginning and other communication events later on that show progress. But there are also suspicious trajectories: a large amount of clarification late in the iteration, perhaps

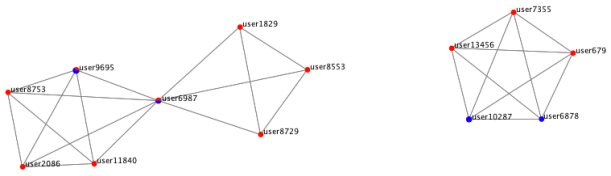

Figure 3: Example of a social network

even after the issue seemed to be solved. Or no clarification at all, even though the issue seems to be complex.

B. Are there any communication breakdowns?

After identifying those hotspots, the manager most likely wants to continue with a closer investigation. Often, he or she will investigate who participates in a discussion of an issue and who is not. V:ISSUE:LIZER helps managers in this scenario by creating social networks for an issue or a set of issues on the fly. More over, V:ISSUE:LIZER also integrates information of the automatic analysis of online communication into these social networks, i.e. showing for each actor the percentage of clarification and other communication. Figure 3 shows an example of an social network for the issue presented in Figure 2. The developers are presented as nodes (here: anonymized), and connections between nodes are weighted by the amount of communication both developers share about a given issue in a specific time interval. In the example above, the manager might conclude that there is no single person who is coordinating the work around this issue, because there is no actor who participates in all relevant time intervals. A suitable action might be to assign the responsibility to a more experienced developer.

C. Who is knowledgeable about a given topic?

Integrating the right persons in the loop for an important feature is a crucial ability for managers. To support managers in this task, V:ISSUE:LIZER distinguishes between two types of knowledge: domain knowledge that shows in communication events related to clarification and technical knowledge that shows in other communication events. In order to leverage the power of V:ISSUE:LIZER for this task, the manager selects a number of issues that are related to a given topic. V:ISSUE:LIZER integrates the social networks of these issues in a single large network (see Figure 4). The manager looks for candidates with a balanced percentage of clarification and other communication and a medium centrality, actors with high centrality might already have a very high workload.

V. PRELIMINARY EVALUATION

RE Paper

IBM Interviews

VIATEC Software Management Round Table

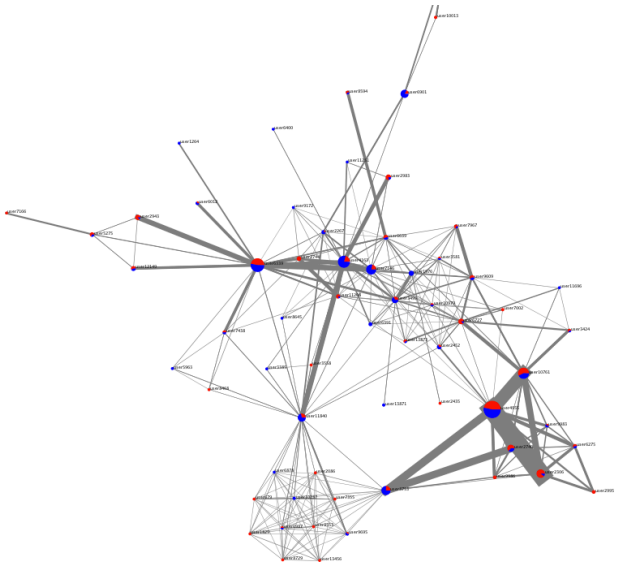

Figure 4: Example of a social network of a set of issues

VI. CONCLUSION AND FUTURE WORK

Write conclusion

Future: Relate Features of Online Communication (i.e. network centrality, late clarification, no clarification) with problems.

TODO LIST

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] C. Treude, P. Gorman, L. Grammel, and M.-A. Storey, "Workitemexplorer: Visualizing software development tasks using an interactive exploration environment," in *Proc. of the 34th Intl. Conf. on Software Engineering (ICSE'12)*, Zurich, Switzerland, 2012, pp. 1399–1402, formal research dem.

[5] R. E. Kraut and L. A. Streeter, "Coordination in software development," *Commun. ACM*, vol. 38, no. 3, pp. 69–81, 1995.

[6] C. Treude and M.-A. Storey, "Awareness 2.0: Staying aware of projects, developers and tasks using dashboards and feeds," in *Proc. of the 32th Intl. Conf. on Software Engineering (ICSE'10)*, vol. 1, Zurich, Switzerland, 2010, pp. 365–374.

[7] J. B. Ellis, S. Wahid, and W. A. Kellogg, "Task and social visualization in software development: Evaluation of a prototype," in *Proc. of the Conf. on Human Factors in Computing Systems (CHI'07)*, San Jose, USA, 2007, pp. 577–586.