Using Developer Conversations to Resolve Uncertainty in Software Development: A Position Paper

Ahmed Shah Mashiyat Michalis Famelis Rick Salay Marsha Chechik
University of Toronto
{mashiyat, famelis, rsalay, chechik}@cs.toronto.edu

ABSTRACT

Software development is a social process: tasks such as implementing a requirement or fixing a bug typically spark conversations between the stakeholders of a software project, where they identify points of uncertainty in the solution space and explore proposals to resolve them. Due to the fluid nature of these interactions, it is hard for project managers to maintain an overall understanding of the state of the discussion and to know when and how to intervene. We propose an approach for extracting the uncertainty information from developer conversations in order to provide managers with analytics. Using these allows us to recommend specific actions that managers can take to better facilitate the resolution of uncertainty.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.8 [**Software Engineering**]: Metrics—*Process metrics*

General Terms

Project management, software analytics

Keywords

Uncertainty management, natural language processing

1. INTRODUCTION

Software development is an inherently collaborative process. Stakeholders typically engage in conversations to manage, coordinate, debate and elaborate different tasks such as implementing enhancements, fixing bugs, etc. In such conversations, people often express their uncertainty about different artifacts in the problem and solution spaces. We call these utterances "points of uncertainty" (PoU). These appear for reasons such as lack of knowledge about a requirement, conflicting opinions regarding the problem, different solution alternatives, etc.

Since human interactions tend to be fluid, it is hard for managers to maintain a good overall understanding of the development of a given conversation, namely, questions raised, solutions proposed, if any, and a current state of accepting or rejecting them. This may mean that a question can remain unanswered or a proposal be adopted prematurely, both of which can damage the quality of the resulting product. Similarly, discussions may stagnate, either because the relevant stakeholders have not been invited or because decisions about proposed solutions have not been made, adversely impacting efficiency of the development team.

In this paper, we propose an approach for identifying PoUs, tracking uncertainty resolution, and effectively managing it, by providing managers with analytics about conversations and recommending actions in situations where intervention could be beneficial. We propose to automatically infer points of uncertainty from sources that stakeholders/developers already produce as part of their collaboration – conversation logs captured by social computing technologies such as team management tools, forums, etc. The rest of the paper is organized as follows: in Sec. 2, we introduce a motivating example. We present our approach in Sec. 3. We discuss related work in Sec. 4 and conclude in Sec. 5.

2. MOTIVATING EXAMPLE

Fig. 1 shows a snippet of a real conversation from `jazz.net` between developers working on a enhancement ticket. The discussion is centered around how new Test Execution Records (TERs) for a Test Plan (TP) should be generated from existing ones contained in a TER list view, for a new round of testing. The conversation consists of a series of entries, each having a unique author and timestamp. Each entry consists of a comment and/or an update to the status of the ticket (open, resolved, etc.). We have changed the names of authors and omitted irrelevant entries.

Bao opens the ticket in entry 0. The first comment comes some months later, when Ali asks two questions in entry 1. In Fig. 1, we have separated entry 1 into two sub-entries, one for each individual utterance. Entry 1a contains the first question which asks whether all selected TERs should belong to the same TP. Entry 1b contains the second question which asks whether new TERs should inherit information from existing ones. We refer to questions like these as *points of uncertainty* (PoUs). After a few days of inaction, Bao invites a third party, Charles, to the conversation (entry 2). Charles proposes solutions to questions 1a and 1b in the sub-entries 3a and 3b respectively. Later (entry 5), Bao comments on Charles's proposals as follows: in entry 5a, he

0: Bao (Feb 29, 2012, 10:00:37 AM)	
	Task status: Open

1: Ali (Jun 27, 2012, 1:07:47 PM)	
a	... Would all of the TERs that you want to select belong to the same test plan? ...
b	Would you also expect that the newly generated [TERs] would inherit all other information from the [TER] that it corresponds to? ...

2: Bao (Jul 11, 2012, 2:19:20 PM):	
	@Charles can you please provide input for @Ali question in comment 1?

3: Charles (Jul 11, 2012, 2:50:08 PM)	
a	Yes, in the use case I was considering, they would all be for the same test plan.
b	... It would be nice if we could be exible and provide users with options. ... They could infer that if they want to get more control over the overriding of the elds (e.g., for some [TERs] they would want one value, for others another value, etc.) then they would just have to select a smaller set of [TERs] to generate from.

• • •

5: Bao (Jul 11, 2012, 2:50:08 PM)	
a	All the selected [TERs] should be part of same same test plan.
b	Fields like Environment, Priority, ... will be copied from existing [TERs]. ...
c	No extra dialog will be provided to edit them before copy. ...

• • •

13: Bao (Jul 18, 2012, 8:53:51 AM)	
	Task status: Open→Resolved

14: David (Jul 19, 2012, 2:53:31 PM)	
a	Task status: Resolved→Reopened
b	...did you realize that once a test plan is approved you can no longer [generate] execution records for it through the Test Plan editor?...

• • •

20: Ali (Jul 24, 2012, 11:46:02 AM)
Task status: Reopened→Resolved

• • •

23: Elli (Aug 20, 2012, 5:55:16 PM)	
a	Task status: Resolved→Verified
b	...I though there was a permission to allow a test lead or other such role to add TER's to an approved TP?

• • •

26: Charles (Sep 17, 2012, 1:24:35 PM)	
	...in comment 23, you indicate a permission but did you intend to mean a process behavior?

27: Fran (Oct 5, 2012, 9:52:56 AM)	
	Task status: Verified→Closed

Figure 1: A snippet of a conversation between developers. Entries are broken down to individual utterances. Square brackets indicate where the fragments have been edited for clarity.

agrees with Charles proposal for question 1a, expressed in in entry 3a. But in entry 5c, he disagrees with Charles's proposal in 3b and instead proposes an alternative solution in entry 5b. Following this, Bao and Charles engage in a discussion (not shown in Fig. 1, debating and elaborating the different proposals. By entry 13, the three have reached decisions for the questions posed in 1a and 1b, and Bao closes the ticket. However, in entry 14, David joins the discussion and objects to flagging the ticket as "resolved". After reopening the ticket (entry 14a), David poses a new question (entry 14b) about a previously unexamined aspect of the problem. In the discussion that follows, David's concerns are addressed (i.e., the new PoU is *resolved*) and the ticket is marked as resolved in entry 20. A month later, Elli changes the status to "verified" and asks a new question (entry 23b). This question is not answered and prompts a second question in entry 26 by Fran. In entry 27, the ticket is marked as "closed", while both questions remain unanswered.

This conversation allows us to motivate several parts of our approach. Assume that Gul is a manager responsible for the component under discussion. We want to provide Gul with analytics and action recommendations for this conversation. Below we propose an indicative, non-exhaustive list of five use cases of recommendation actions:

R1: We notice that the questions asked by Ali on June 27th (entry 1) did not receive any attention until July 11th (entry 2). It would be useful for Gul to know *which questions have not been answered for longer than some desired period of time T*. That way, Gul could have intervened to ensure that Ali's question got answered in time.

R2: When questions are not answered, it causes inefficiencies and delays in the development process. However, if they remain unanswered when the conversation ends, this could also impact the quality of the produced software. In our example, the questions posed by Elli and Fran in entries 23 and 26 are left unanswered when the ticket is closed in entry 27. We want to report *which questions remain unanswered at the end of a conversation*, allowing managers to assess the importance of the remaining questions and decide which require intervention, thus ensuring better software quality.

R3: For Gul, it would be useful to know *for which questions more than one solutions was proposed or which have "mature enough" proposals*, in order to step in and ask the participants to decide whether an answer has been obtained. For example, after entry 5a, the majority of participants in the conversation have weighed in regarding the question in 1a: In entry 3a, Charles explicitly replies with *"Yes"* and in entry 5a Bao also agrees. This could be an indication for Gul to step in and ask Ali whether the answer given is satisfactory, thus resolving this issue. Similarly, by the time of entry 5b, two alternative proposals, expressed in entries 3b and 5b, have been laid out for addressing the question in entry 1b. Gul could decide to intervene and ask participants for their opinions about the two proposed alternatives.

R4: Based on the evolution of the discussion, we can notice that Bao's decision to mark the ticket as "resolved" in entry 13 was premature. This prompted David join the conversation, re-open the ticket and ask a new question (entry 14b). If David had been invited to join the conversation at an earlier point than entry 13, the decisions made by the other participants (e.g., the solutions to questions in entry 1) before closing the discussion might have been different. It would thus have been useful for Gul to be able to *identify open issues which are considered settled either too soon or without the participation of all of the relevant stakeholders.*

R5: Finally, *when an issue is re-opened, it is useful to remember the alternative resolutions that were considered in the past.* That way, the participants of the discussion do not have to re-brainstorm all possibilities from scratch. In our example, when David re-opens the ticket (entry 14), questions that were considered settled (such as the ones in entry 1) may need to be re-examined in the light of the newly raised concern. Assuming Bao's proposal in entry 5b was selected as the solution to the question in 1b, when David re-opens the issue (and thus the previously settled questions), it is useful to be aware of the (rejected) proposal made by Charles in entry 3b, since the new concerns voiced by David may now make it acceptable.

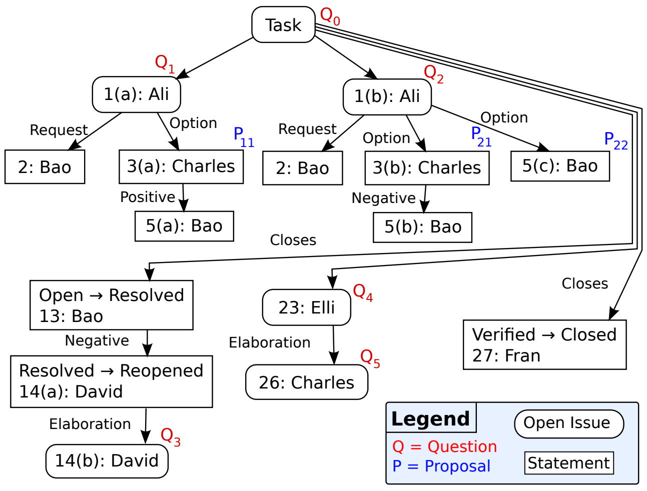


Figure 2: TAS model of the conversation in Fig. 1.

3. APPROACH

In this paper, we propose an approach for managing the resolution of uncertainty in developer conversations such as the one in Fig. 1. Our approach consists of the following steps: (a) Mine the natural language conversation log to create an intermediate representation model of its structure. Using this representation, we can identify points of uncertainty in the conversation as well as the proposals made to address them. (b) Use the mined model to create an analytics dashboard. The metrics displayed in the dashboard are generated using a quantitative theory over the intermediate representation. (c) Use the intermediate representation and the metrics to generate recommendations for meaningful actions that managers should take at various points during the discussion. In the rest of the section, we discuss these steps.

3.1 Mining Conversations

Our approach requires us to identify the correlation between different parts of the ("free form") conversation and to identify what parts of the text are utterances representing points of uncertainty (PoUs). Such an utterance is a particular linguistic expression that contains some uncertainty (e.g., contains words like "maybe", "probably", "not sure", etc.). Different utterances may refer to the same PoU; thus, a PoU is an abstract construct. To determine which PoU a given utterance refers to, we need to identify parts of the conversation representing questions (Qs) and parts representing proposals to answer such questions (Ps).

We plan to use natural language processing techniques, such as text segmentation and tagging approaches [6], to identify these PoUs in a conversation. We will then construct an argumentation model based on the Twente Argument Schema (TAS) [8] using these. A *TAS argumentation model* expresses the structure of a conversation at a higher level of abstraction in order to reveal the questions posed and proposals to answer these questions. TAS identifies the relationships between different segments of a natural language conversation (e.g., whether the segments agree, disagree, provide an elaboration, etc.), preserving the coherence and flow of the conversation.

A TAS model is a tree structure that can be read left to right and top to bottom to understand the conversation transcript in a structured way [8]. In Fig. 2, we show the tree that results from applying the TAS schema for the relevant parts of the example of Fig. 1 to identify the questions (Oval, marked with Q_i), the proposals (Rectangle, marked with P_i), and their correlations. These are indicated by an arrow and by numbering, e.g., the two proposals for Q_2 are marked as P_{21} and P_{22}. Fig. 2 shows the tree for the fragment of the entire conversation from `jazz.net` which is shown in Fig. 1, but a complete tree can be constructed as well. Questions in this model correspond to OpenIssues in the TAS schema. Statements are represented using rectangles and those which are proposals are marked with P_{ij}. *Option* is used if the child is a possible answer, option or solution w.r.t. the parent. We extended the TAS schema with constructs such as *Part of*, *Closes*, etc. to represent the type of conversations we are interested in.

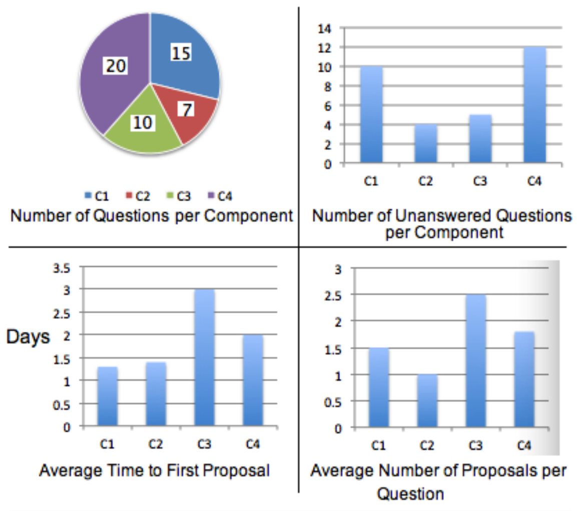
For the example in Sec. 2, the root node in Fig. 1 is the entry 0 corresponding to the question Q_0. With entry 1, Ali introduces questions 1(a) and 1(b). After two weeks, entry 2 is made where Bao intervenes and asks Charles to comment on these questions (denoted "2: Bao"). With entry 3, Charles agrees to the first part of the question *1(a)* (3(a): Charles) and proposes an alternative to the second part of the question *1(b)*. When entry 5 is made, Bao also agrees to the proposal for question *1(a)* (5(a): Bao) (Positive) disagrees with the proposal of *3(b)* (Negative) and puts in a counter proposal (5(c):Bao).

3.2 Uncertainty Analytics

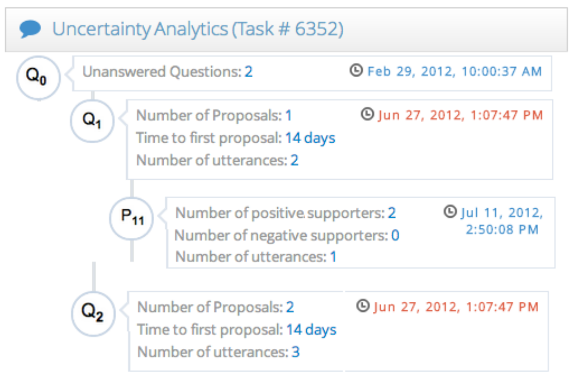
Our objective with analytics is to define metrics that can support the production of recommendations like **R1-R5** in Sec. 2. Two kinds of uncertainty analysis are possible: *online* – an analysis of an in-progress conversation up to the current time in order to facilitate the discussion, and *offline* – an analysis of the entire conversation after it completes in order to learn lessons that may improve future development projects. In this paper, we focus on online analysis.

The most natural metrics for our purposes are ones related to the resolution of a point of uncertainty (PoU) such as, "how many unresolved PoU's are there?","how long does a PoU take to resolve", etc. To compute any of these, we need a way of determining whether a PoU has been resolved at a given time t, that is, for each question q we want to identify the exact point in the discussion where the group decides to accept a particular proposal that answers q. Unfortunately, determining this is challenging for several reasons:
It may be difficult to determine who should be part of the discussion. Each discussion is made up of an ad-hoc group of people. How many and who are needed for a decision to be considered "final"?
It may be difficult to determine the stance an individual has regarding a particular proposal. It is not necessary for individuals to explicitly verbalize their stance regarding a proposal like Charles does in 3(a) (see Fig. 1). For example, Bao in 5(b) elaborates and thus lends support to Charles' proposal without using words that could be understood as an explicit approval.
It is not clear what the absence of an explicit expression of approval, disapproval or indifference signifies for each participant Demographic, cultural, etc. differences between individuals result in different default behaviors. For example, some individuals may choose to not speak unless they explicitly disagree with a point. Others may not discuss one proposal because they have not yet given it enough thought.

(a)



(b)


Uncertainty Analytics (Task # 6352)

Q_0 Unanswered Questions: 2 — Feb 29, 2012, 10:00:37 AM

Q_1 Number of Proposals: 1 — Jun 27, 2012, 1:07:47 PM
Time to first proposal: 14 days
Number of utterances: 2

P_{11} Number of positive supporters: 2 — Jul 11, 2012, 2:50:08 PM
Number of negative supporters: 0
Number of utterances: 1

Q_2 Number of Proposals: 2 — Jun 27, 2012, 1:07:47 PM
Time to first proposal: 14 days
Number of utterances: 3

Figure 3: An uncertainty analytics dashboard: (a) a top-level view; (b) a drill-down view.

Given that all we have is the recorded conversation, how do we differentiate between these two behaviours?

Due to these challenges, we have chosen to instead use metrics that are simple computations based on the structure and content of the TAS model, instead of trying to determine when a PoU is resolved. We describe these metrics below.

Metrics and Dashboard. In Fig. 3.2, we show our vision of how an uncertainty dashboard may look from a user's (e.g., manager's) perspective. The dashboard gives the user an easy overview of the questions and proposals within a particular component. We use the term 'component' loosely to mean different a decomposition unit of the project, such as a module, feature, etc. To get more detail, the user can "drill down" into a component or into specific PoU's.

We propose to use the following metrics:

Number of unanswered questions. A question is considered unanswered if it has no proposals. Thus, this metric is computed as the number of OpenIssue nodes without an Option child.

Number of proposals per question. This metric is computed as the number of Option children of the OpenIssue node for the question.

Time to first proposal. This metric is computed using the time stamp of the first utterance that introduces a proposal for a given question.

Number of positive/negative supporters for a proposal. This metric is computed as the number of children nodes of an proposal node (i.e., an Option child of an OpenIssue) connected by a Positive or Negative relation. We assume that a proposal node always has an implicit count of "one positive".

Number of utterances per question/proposal. This metric is computed as the number of children nodes for an OpenIssue node (for a question) or one of its Option child nodes (for a proposal).

The various widgets on the dashboard show aggregate values for metrics computed from the TAS model for the conversation. The top left widget of (Fig. 3.2(a)) shows the number of questions raised within each decomposition units (e.g., C1 has 15 questions, C4 has 20 and so on); the top right widget shows the number of unanswered questions within each decomposition units (e.g., C2 has four). The bottom right widget reports on the average proposals per question (C3 has an average of 2.5 proposals). Finally, the bottom left widget reports on the average time before a question gets answered (in C3, it is 3 days). Clicking on an overview displays more detailed information. Fig. 3.2(b) shows the metrics for the conversation described in Sec. 2 and represented in Fig. 2. For example, it indicates that task Q_0 has two questions without proposals. Q_1 in Fig. 2 has a single child proposal (represented as an Option) and two utterances. P_{11} has one utterance and one positive response by Bao. Since Q_2 has two Option children in Fig. 2, it thus has two proposals.

3.3 Recommending Actions

In Sec. 2, we introduced five recommendation actions (**R1**-**R5**) for managers. Here, we demonstrate how these recommendations can be generated using the metrics computed to populate the uncertainty analytics dashboard in Sec. 3.2.

R1: Based on the answer to *which questions have not been answered for longer than some desired period of time T?*, we want to generate a recommendation to the manager to intervene and request that relevant stakeholders weigh in. We can find such open questions using the metric "number of unanswered questions" and filtering out those that were filed after the cut-off time period. In our example, and assuming that the manager Gul has set the period T to be 2 days, the system would generate such recommendations 2 days after entry 1 was posted. Gul could then step in and solicit replies as early as June 29th, as opposed to July 11th when entry 2 was posted.

R2: This action is based on the answer to *which questions remain unanswered at the end of a conversation?*. This can be determined similarly to **R1**, but when the conversation is indicated as finished. We find unanswered questions by looking at the metric "number of unanswered questions" filtering out questions from ongoing conversations. In our example, after entry 27, when Bao closes the ticket, Gul will be prodded to reopen the conversation since the questions posed in entries 23 and 26 remained unanswered (see Fig. 2).

R3: This action requires answering *for which questions more than one solution was proposed or which have "mature enough" proposals*. For such questions, the system would recommend the manager to ask the participants whether they are ready to make a decision, recapping the proposed solutions, if necessary. Finding questions with many proposals can be done with the *number of proposals per question* metric. The maturity of proposals can be assessed using the metrics *number of positive/negative supporters for a proposal* and *number of utterances per question/proposal*. High values for these metrics indicate that the proposal has been debated in depth and so may be considered mature. In our example, at the time when entry 5 is made, the system would recommend to Gul to ask participants whether they want to accept the proposal made by Charles in entry 3a as a solution for the question in entry 1a since it has the support of two participants. Since there is only one proposal, Gul could ask participants whether they want to brainstorm some more. At the same time, the system would generate a recommendation to ask whether the participants are ready to make a decision regarding the question in entry 1b, since it has generated discussion and proposals.

R4: Finding *which open issues are considered settled either too soon or without the participation of all the relevant stakeholders?* we can generate a recommendation to reopen the conversation and invite potentially interested stakeholders. The relevant metrics are *time to first proposal* and *number of utterances per question/proposal*. An issue with low values for these metrics indicates that the resolution might be immature. In our example, at the time the conversation is marked as "resolved" at entry 13, the system should recommend to Gul to re-open it, inviting more dialog and potentially more people, such as David, into the discussion.

R5: Finally, in the cases of an issue being reopened, we want the system to remind managers *what alternative proposals were considered about it in the past*. For this, we need to maintain the TAS representation of the entire conversation. When a question is re-opened, the system can recommend to the manager to remind the participants of previously rejected proposals. The rationale why they were rejected can also be re-instated from the TAS model. In our example, when David reopens the conversation in entry 14, our system would recommend to Gul to remind participants of the alternative proposals to the questions 1a and 1b. This recommendation can take the form of the appropriate TAS subtree, i.e., the subtrees under 1a and 1b in Fig. 2.

4. RELATED WORK

The works on tracking uncertainty [5] and clarification and coordination in developer conversations [4] are the closest to ours. In [5], evidence theory is used to keep track of uncertainty as it increases or decreases during information fusion. This work is more oriented to the real-time fusion of sensor data rather than natural language text streams such as conversations. In [4], different types of requirements clarification communication have been captured and analyzed over time to produce visual analytics for managers and other stakeholders. Our work differs from this research by using an argumentation model as an intermediate step in the analytics process. The model provides a way to reason about the points of uncertainty and hence provides a more general platform on which to base uncertainty analytics.

Identifying the points of uncertainty in a conversation has been studied in different contexts. For example, [9] uses a machine learning algorithm called Conditional Random Fields (CRFs) to identify uncertainty cues, enabling automatic identification of speculative requirements expressed in natural language in requirements documents. Similar work has been done to identify uncertainty in linguistic data [1], or during information extraction [3]. We hope to exploit techniques such as these in the conversion of conversations into TAS models.

In our work, we used TAS as a formal intermediate representation of the stakeholders' conversations. Rhetorical Structure Theory (RST) [7] would have been another strong contender: RST explains coherence of a sentence by suggesting a hierarchical, connected structure of the text, in which every part of the text has a *role* – its function w.r.t. the other parts of the text. We chose TAS because it most closely fits the problem of representing the points of uncertainty within a text.

Our previous work on representing and reasoning with uncertainty also focused on the manual annotation of design artifacts to indicate points of uncertainty [2]. The current work addresses another aspect of uncertainty in a development process. In the future, we hope to automatically produce such annotations using the TAS model and traceability links to the design artifacts that the developer conversations are talking about. This will provide richer opportunities for reasoning with and resolving uncertainty in the project.

5. CONCLUSION

We have presented an approach for managing uncertainty in developer conversations. We propose to capture utterances of uncertainty in an intermediate representation based on the Twente Argument Schema (TAS). Using the TAS model, we can compute metrics that allow us to generate recommendations for managers to take actions, as well as to provide them with an uncertainty analytics dashboard.

In the future, we intend to address the challenges in realizing this approach, e.g., (a) How to best automate the creation of the TAS model? (b) How to identify the resolution of uncertainty? (cf. Sec. 3.2) (c) What are the best metrics, analytics and action recommendations to provide to managers? (d) How can we use the semantics of the utterances to draw connections with other relevant organizational, social and software models?

6. ACKNOWLEDGMENTS

Many of the ideas in this paper are based on conversations with Daniela Damian, Eric Knauss, and Jennifer Horkoff.

7. REFERENCES

[1] A. Auger and J. Roy. Expression of Uncertainty in Linguistic Data. In *Proc. of ICIF'08*, pages 1–8, 2008.

[2] M. Famelis, Rick Salay, and M. Chechik. Partial Models: Towards Modeling and Reasoning with Uncertainty. In *Proc. of ICSE'12*, pages 573–583, 2012.

[3] B. Goujon. Uncertainty Detection for Information Extraction. In *Proc. of RANLP'09*, 2009.

[4] E. Knauss and D. Damian. V: ISSUE: LIZER: Exploring Requirements Clarification in Online Communication Over Time. In *Proc. of ICSE'13*, pages 1327–1330, 2013.

[5] H.J. Steinhauer, A. Karlsson, and S.F. Andler. Traceable Uncertainty. In *Proc. of FUSION'13*, pages 1582–1589, 2013.

[6] M. Strohmaier. Purpose Tagging: Capturing User Intent to Assist Goal-oriented Social Search. In *Proc. of SSM'08*, pages 35–42, 2008.

[7] M. Taboada and W. C. MANN. Rhetorical Structure Theory: Looking Back and Moving Ahead. *J. Discourse Studies*, 8:423–459, 2006.

[8] D. Verbree, R. Rienks, and D. Heylen. First Steps Towards the Automatic Construction of Argument-Diagrams from Real Discussions. In *Proc. of COMMA'06*, pages 183–194, 2006.

[9] H. Yang, A. De Roeck, V. Gervasi, A. Willis, and B. Nuseibeh. Speculative Requirements: Automatic Detection of Uncertainty in Natural Language Requirements. In *Proc. of RE'12*, pages 11–20, 2012.