# Canary: Extracting Requirements-Related Information from Online Discussions

Georgi M. Kanchev
Lancaster University
g.kanchev@lancaster.ac.uk

Pradeep K. Murukannaiah
Rochester Institute of Technology
pkmvse@rit.edu

Amit K. Chopra
Lancaster University
amit.chopra@lancaster.ac.uk

Pete Sawyer
Aston University
p.sawyer@aston.ac.uk

*Abstract*—**Online discussions about software applications generate a large amount of requirements-related information. This information can potentially be usefully applied in requirements engineering; however currently, there are few systematic approaches for extracting such information. To address this gap, we propose Canary, an approach for extracting and querying requirements-related information in online discussions. The highlight of our approach is a high-level query language that combines aspects of both requirements and discussion in online forums. We give the semantics of the query language in terms of relational databases and SQL. We demonstrate the usefulness of the language using examples on real data extracted from online discussions. Our approach relies on human annotations of online discussions. We highlight the subtleties involved in interpreting the content in online discussions and the assumptions and choices we made to effectively address them. We demonstrate the feasibility of generating high-quality annotations by obtaining them from lay Amazon Mechanical Turk users.**

*Keywords*-**Requirements elicitation; Crowdsourcing; Social media; Online discussions; Query language**

## I. Introduction

Conventional methods for requirements elicitation involve direct communication between stakeholders and requirements engineers via interviews, questionnaires, focus groups, workshops, and consultations with field experts [25]. Combinations of such methods are often successfully used, but eliciting the more elusive, tacit requirements remains challenging [34]. Informing requirements from user feedback on applications [31], [38], [18] and *crowdsourcing* requirements [12], [23] are new avenues for obtaining a fuller picture of user requirements.

This paper applies crowdsourcing to support a rich, dynamic, and user-driven understanding of an application's requirements from information in online discussion forums. Further, it employ argumentation [29], [10], [24] to structure discussions for obtaining high-quality requirements.

An online discussion about a software application typically has the following structure. First, a user posts a question about how he or she may accomplish something with a software application, or describes a feature that he or she wishes the application had. We consider such natural language descriptions—problems the users encounter and descriptions of what they expect the application to do—as requirements, broadly. Next, other users may respond with expressions of support for the user's comment or rebut it, e.g., by pointing out the infeasibility of what the user is requesting. Yet others may respond by proposing solutions that accomplish what the original user wanted. That is, a solution describes an existing means of addressing (solving) a requirement. In addition, users may employ upvotes and downvotes to indicate their support (or lack of support) for comments. In general, such a discussion would have a nested structure in that users may respond to any user's comment, not just the original poster's.

In a nutshell, online discussions include two kinds of information valuable to RE: requirements-oriented and social interaction or argumentation-oriented. Extracting such information helps understand the challenges stakeholders face, which in turn helps formulate requirements and prioritize development tasks. However, current techniques are inadequate for extracting and utilizing information in online discussions.

Several aspects of online discussions make the task of using them to inform application development challenging. One, the data is often voluminous with a single top-level post often invoking a long discussion. Two, most of the discussion is carried out in natural language; so identifying aspects of the discussion (e.g., whether some comment expresses a requirement) is nontrivial. Three, there is currently no systematic methodology of leveraging requirements-related information from social media that supports features related to social interaction. For example, there is no tool that would allow a requirements engineer at Google to formulate a query such as *give me the five most controversial requirements about Google Maps over the last two months*, where "controversy" captures aspects of social interaction (discussion). Our contributions in the paper lie in addressing these challenges.

*Contributions:* Our overarching contribution is a methodology called Canary that effectively creates a requirements-oriented view of online discussions as Figure 1 shows. The centerpiece of the methodology is a conceptual model that combines requirements-relevant and argumentation-relevant aspects of online discussions. We demonstrate that it is feasible to transform online discussions into instances of this model by crowdsourcing annotations of the discussions. The payoff of Canary is a novel high-level query language that combines features of requirements and argumentation in interesting ways and whose queries can be executed over databases of the annotated discussions. For practical usefulness, we provide the semantics of Canary queries in terms of SQL queries. We have implemented a prototype and demonstrate the results on databases of real online discussions. Canary makes writing a

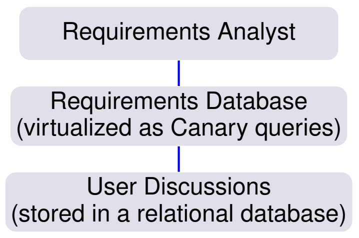

Requirements Analyst
Requirements Database
(virtualized as Canary queries)
User Discussions
(stored in a relational database)

Fig. 1. Canary realizes a requirements-oriented store over user discussions stored in traditional information stores. The view is realized via a mapping from Canary queries to SQL queries.

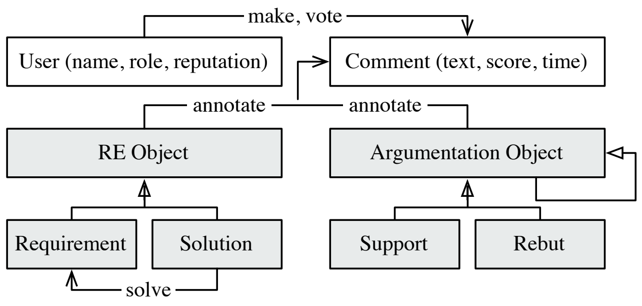


Fig. 2. Conceptual model of information considered in Canary

variety of sophisticated queries simple, which would otherwise be written in a low-level language. Importantly, it goes significantly beyond technology such as IBM DOORS [13], where querying is limited to text searches.

Our contributions are informed by considerations of simplicity and the structure of online discussions. Unrestricted (free-form) interactions in online discussions can be quite rich and subtle in meaning. We kept our models of both requirements and argumentation simple. However, we exploit the structure and information in online discussions to our advantage. Thus, e.g., when a requirement appears in a reply to a comment which is also a requirement, we consider the former a *derived* requirement. We also exploit discussion features such as votes and user reputation in computing queries.

*Organization:* Section II describes the structure and subtleties we observed in online discussions and motivates our conceptual model and methodology. Section III illustrates running queries in Canary and its formal syntax, semantics, and implementation. Section IV evaluates the efficacy of obtaining annotations from lay users. Section V positions our contributions with respect to the literature. Section VI discusses our contributions and future directions.

## II. Methodological Details

We present a conceptual model of online discussions and describe the subtleties in real discussions that we encountered.

### *A. Conceptual Model*

Our conceptual model relates elements of users discussions with elements of requirements and argumentation. Figure 2 shows main types of information Canary considers.

User discussions capture information related to social interaction between application *users*, who may be playing different *roles* in the discussion. Interaction is captured via *comments* (and their replies) and users' *votes* for comments, usually measured in a metric called *score*. Requirements information is captured via annotations to comments in the user discussion. Currently, Canary supports two kinds of requirements annotations, *requirement* and *solution* for a requirement (a solution always refers to a requirement). A requirement may have multiple solutions. Argumentation information is captured via annotations to comments made in response to a requirement or solution. The two kinds of argument annotations currently supported in Canary are *support* and *rebuttal*. A requirement or a solution may have multiple support and rebuttal comments. An argument comment may itself be argued about; thus, argumentation is unbounded in depth.

### *B. Canary Methodology*

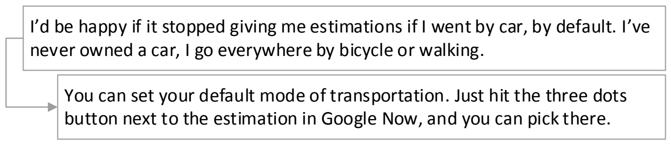
Canary has four main steps, potentially performed by a requirements analyst.

1) *Acquire discussions.* Analyst acquires data from online discussion forums, where possible by using an API (e.g., from Reddit). The extracted data reflects the discussion accurately, including the flow of the discussion, user names, reputation, votes, and so on.
2) *Acquiring annotations.* At this stage, the discussion data contains no information about requirements or argumentation-related information. To obtain such information, the analyst sets up annotation tasks on Amazon Mechanical Turk (MTurk).
3) *Creating a database of discussions.* The analyst creates a relational database that reflects the schema of Figure 2 (the full schema is available in [5]) and loads the annotated discussion data into the database.
4) *Run Canary queries.* Analyst runs Canary queries against the database. Internally, the Canary compiler generates the appropriate SQL queries that can be run on the database.

### *C. Challenges and Assumptions*

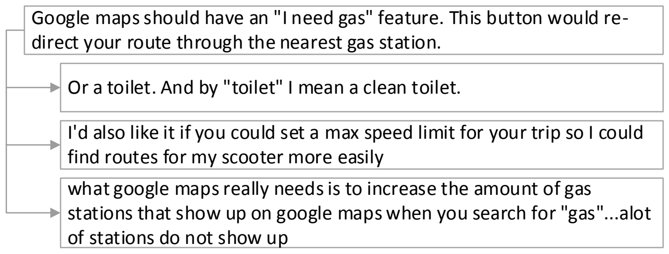
Online forums provide a flexible way of interaction between users. This creates several non-trivial challenges for acquiring requirements-related information from them. One primary challenge is how do we infer relationships between annotated comments (objects), e.g., for the purpose of determining whether the sentiment for some requirement should count as sentiment toward another or whether the solution for one should count as solution for another. To be precise, the challenge arises from the fact that (1) there are no restrictions on what a user can say at any point in the discussion and (2) that discussion is of the form of a tree of unbounded depth.

In general, our strategy is to infer relationships between two objects only if one of them appears in a comment that is a reply (however deeply nested) to the comment in which


I'd be happy if it stopped giving me estimations if I went by car, by default. I've never owned a car, I go everywhere by bicycle or walking.

You can set your default mode of transportation. Just hit the three dots button next to the estimation in Google Now, and you can pick there.

Fig. 3. Example of a solution to a requirement

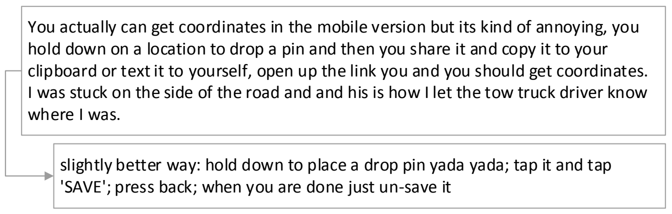

Google maps should have an "I need gas" feature. This button would re-direct your route through the nearest gas station.

Or a toilet. And by "toilet" I mean a clean toilet.

I'd also like it if you could set a max speed limit for your trip so I could find routes for my scooter more easily

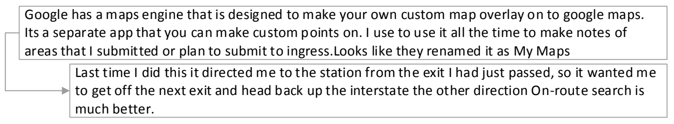
what google maps really needs is to increase the amount of gas stations that show up on google maps when you search for "gas"...alot of stations do not show up

Fig. 4. Example of derived requirements


You actually can get coordinates in the mobile version but its kind of annoying, you hold down on a location to drop a pin and then you share it and copy it to your clipboard or text it to yourself, open up the link you and you should get coordinates. I was stuck on the side of the road and and his is how I let the tow truck driver know where I was.

slightly better way: hold down to place a drop pin yada yada; tap it and tap 'SAVE'; press back; when you are done just un-save it

Fig. 5. Example of a derived solution


Google has a maps engine that is designed to make your own custom map overlay on to google maps. Its a separate app that you can make custom points on. I use to use it all the time to make notes of areas that I submitted or plan to submit to ingress.Looks like they renamed it as My Maps

Last time I did this it directed me to the station from the exit I had just passed, so it wanted me to get off the next exit and head back up the interstate the other direction On-route search is much better.

Fig. 6. Example of a rebuttal to a requirement

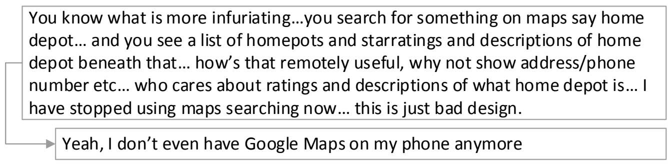
the other one appears. This means that it is possible that we would miss relationships between two objects that do not appear along the same path in the discussion tree. This is the price we chose to pay for simplicity of annotations. Specifically, if we had wanted to capture all relationships regardless of nesting, the annotation task would have become much more difficult. In that case, the annotators would have to give a unique name to each object and indicate explicitly the relationships among them. However, because we decided to forego such relationships, annotators need not use labels or indicate relationships; they simply need to pick the annotation that applies best. We expect that arbitrary relationships would be rare occurrences in any discussion.

The implementation of Canary queries uses *propagation* to infer relationships among annotated objects that arise from nesting in the discussion. We propagate both semantics (e.g., if a solution is deeply nested in a requirement, we assume that the solution addresses the requirement) and sentiment, which is captured as a metric over the number of supports and rebuttals and up and down votes for an object. Propagating sentiment would mean that, e.g., if a requirement acquires some sentiment then its parent requirement (if any) will also acquire that sentiment. In general, a parent would acquire the sentiments of all its children.

In order to support our assumptions we show examples from real discussions of instances where some subtleties occur. When a solution is nested in the responses to a requirement, we assume that the solution is proposed to address the requirement (Figure 3). A requirement can occur in reply to another requirement. In this case, we assume that the former is *derived* from the latter (Figure 4 shows three derived ones). The notion of a derived solution is analogous (Figure 5).

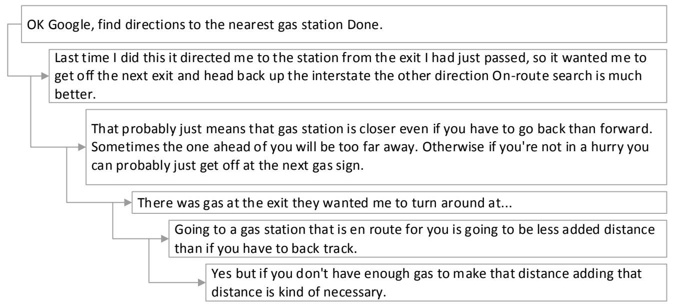
Argumentation objects can be nested as well. Positive and negative argumentation about a requirement is shown in Figure 6 and Figure 7, respectively, and mixed argumentation is shown in Figure 8. A support for an object of interest expresses positive sentiment toward it. Supporting the supporting argument adds positive sentiment to the original object. A rebuttal to an object of interest expresses negative sentiment toward it. A rebuttal to this rebuttal adds positive sentiment toward the original object of interest. In unbounded nesting of rebuttals the sentiment may switch between positive and negative toward the root object. Finally, imagine we observe a support to an object of interest. A rebuttal of this support adds negative sentiment to the original object of interest. If we add another rebuttal then the sentiment becomes positive toward the original object. Analogously, supporting of rebuttals adds to the negative sentiment.

Gaps in nesting can occur. Imagine a comment with no object of interest, and then in a reply to this comment we observe something interesting. Such gaps provide a challenge in the way Canary infers relationships between objects of interest. Canary queries propagate to the end of the discussion tree to make sure all relations between objects are detected.


You know what is more infuriating...you search for something on maps say home depot... and you see a list of homepots and starratings and descriptions of home depot beneath that... how's that remotely useful, why not show address/phone number etc... who cares about ratings and descriptions of what home depot is... I have stopped using maps searching now... this is just bad design.

Yeah, I don't even have Google Maps on my phone anymore

Fig. 7. Example of a support to a requirement


OK Google, find directions to the nearest gas station Done.

Last time I did this it directed me to the station from the exit I had just passed, so it wanted me to get off the next exit and head back up the interstate the other direction On-route search is much better.

That probably just means that gas station is closer even if you have to go back than forward. Sometimes the one ahead of you will be too far away. Otherwise if you're not in a hurry you can probably just get off at the next gas sign.

There was gas at the exit they wanted me to turn around at...

Going to a gas station that is en route for you is going to be less added distance than if you have to back track.

Yes but if you don't have enough gas to make that distance adding that distance is kind of necessary.

Fig. 8. Example of mixed argumentation about a solution

The flexibility of natural language allows for more than one object of interest in the same comment. We make the assumption that each comment contains one object of interest. During annotation we favor RE objects over argumentation.

Online discussion forums can differ. In this paper, we include observations from two forums: Reddit.com (which allows for unbounded nesting of comments) and GoogleForums.com (which only allows one level of nesting). Forums also store different details of the interaction that become available metadata. For example, Reddit stores the sum of votes for each entity in a metric called score. Google forums has a role associated with its users, such as Google community manager or a regular user. Reddit has a numerical value for reputation of user of the forum. Queries for missing metadata, for example a query for user role in Reddit, are handled gracefully by Canary. It runs the rest of the query as normal and ignores the condition, producing a warning.

## III. Queries in Canary

In this section, we provide examples of high-level Canary queries and their results on a database of real online discussions from Reddit. Then, we provide the formal syntax and semantics of queries. For brevity, we only show an overview of the semantics (full details available at [5]).

### A. Example Discussion

Figure 9 shows an example discussion that illustrates the framework. In the example, users are discussing features and requirements of Google Maps. The example is extracted from Reddit. John suggests a requirement about being able to save addresses in Google Maps. The requirement gets a score of 805. Mary expresses support for the requirement. Henry and Patrick both propose solutions for the requirement (and get a score of their own, which is a result of summing upvotes and downvotes). Henry's solution attracts rebuttal comments in addition to a score. Patrick's solution attracts support and a score. In general, each comment may attract up and down votes (resulting in a positive/negative score), as explained above.

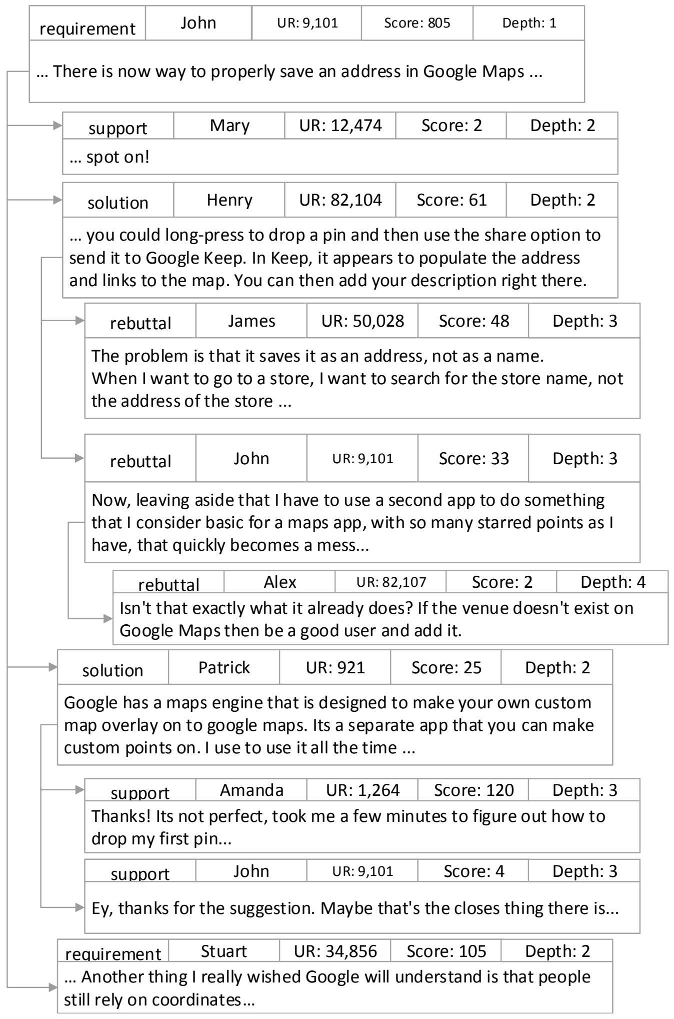
### B. Queries

Below are examples of queries that leverage the framework. Each query is shown in a listing, and the output of the query is shown in a table immediately after. The queries are run on the example discussion shown in Figure 9, so the information returned is taken from there. The natural language text is shortened for space reasons.

Figure 10 shows a query to select all objects of interest annotated as requirements in the discussion.

The score of the "save address" requirement is different from the discussion above because of propagation. In the replies of "save address" there is one support with a score of 2 and one derived (nested) requirement with the score of 105. So, the score of these propagates up to the original requirement and is added to its own score, yielding 912.

Requirements are the only objects of interest that are standalone. A solution must address a requirement, a support must


requirement | John | UR: 9,101 | Score: 805 | Depth: 1
... There is now way to properly save an address in Google Maps ...

support | Mary | UR: 12,474 | Score: 2 | Depth: 2
... spot on!

solution | Henry | UR: 82,104 | Score: 61 | Depth: 2
... you could long-press to drop a pin and then use the share option to send it to Google Keep. In Keep, it appears to populate the address and links to the map. You can then add your description right there.

rebuttal | James | UR: 50,028 | Score: 48 | Depth: 3
The problem is that it saves it as an address, not as a name.
When I want to go to a store, I want to search for the store name, not the address of the store ...

rebuttal | John | UR: 9,101 | Score: 33 | Depth: 3
Now, leaving aside that I have to use a second app to do something that I consider basic for a maps app, with so many starred points as I have, that quickly becomes a mess...

rebuttal | Alex | UR: 82,107 | Score: 2 | Depth: 4
Isn't that exactly what it already does? If the venue doesn't exist on Google Maps then be a good user and add it.

solution | Patrick | UR: 921 | Score: 25 | Depth: 2
Google has a maps engine that is designed to make your own custom map overlay on to google maps. Its a separate app that you can make custom points on. I use to use it all the time ...

support | Amanda | UR: 1,264 | Score: 120 | Depth: 3
Thanks! Its not perfect, took me a few minutes to figure out how to drop my first pin...

support | John | UR: 9,101 | Score: 4 | Depth: 3
Ey, thanks for the suggestion. Maybe that's the closes thing there is...

requirement | Stuart | UR: 34,856 | Score: 105 | Depth: 2
... Another thing I really wished Google will understand is that people still rely on coordinates...

Fig. 9. Example discussion following information framework

**requirement**

| text | annotation | user | UR | score | depth |
|---|---|---|---|---|---|
| save address | requirement | John | 9101 | 912 | 1 |
| coordinates | requirement | Stuart | 34856 | 105 | 2 |

Fig. 10. Canary requirement query

address something, and a rebuttal must rebut something. We will exemplify using solutions below; writing queries for support and rebuttal is analogous.

Figure 11 shows a query to select solutions for requirements that mention 'save address'. This figure shows to interesting elements. First, it shows an example of *conditions*, which can be applied to objects of interest to leverage the associated metadata such as score, natural language text (with support for regular expressions and fuzzy matching), user reputation or role, creation time, and depth in discussion. Second, it shows the value of propagation. In the original discussion 'long press...' has a bigger score than '...overlay engine', but with propagated values the score of '...overlay engine' increases greatly because of its two supporting arguments, while the score of 'long press...' drops from the rebutting arguments.

```
solution (
  requirement where text regexp 'save address'
)
```

| text | annotation | user | UR | score | depth |
|---|---|---|---|---|---|
| long press and send to Keep | solution | Henry | 82,104 | -18 | 2 |
| custom overlay engine | solution | Patrick | 921 | 149 | 2 |

Fig. 11. Canary solution query and results

In Figure 12, we query for *popular* requirements; this is an example of what we refer to as *aggregator* queries. The idea of popular is to select those requirements which caused a high amount of positive interaction from the community. As positive interaction we consider score, supporting arguments, or any object of interest in the reply tree that has positive sentiment toward the original object of interest. Canary is able to propagate support through nested positive interaction entities (support of support, support of derived, etc.)

```
popular ( solution ( requirement where regexp
    'save address' ))
```

| text | annotation | user | UR | score | pop score |
|---|---|---|---|---|---|
| custom overlay engine | solution | Patrick | 921 | 149 | 298 |

Fig. 12. Canary aggregator query and results

Popularity is measured as the ratio of the sums of quantitative data about positive and negative interactions.

$$\text{Popularity Score} = \frac{\sum_{i=0}^{n} \text{PV(i)} \times \text{n}}{\sum_{i=0}^{m} \text{NV(i)} \times \text{m}}$$

where $n$ is the number of children with positive sentiment and PV(i) the number of votes for a given child $i$, and $m$ is the number of children with negative sentiment and NV(i) the number of votes for a given child $i$. In Table I, we present the assumptions we can make based on the ratio.

TABLE I
AGGREGATOR ASSUMPTIONS

| Pop ratio | Aggregator | Assumption |
|---|---|---|
| <1 | Unpopular | Negative interaction has prevalence |
| 1 | Controversial | Balance between positive and negative |
| >1 | Popular | Positive interaction has prevalence |

## C. Formal Syntax and Semantics

In this section, we describe the language formally. Table II defines the syntax of Canary.

The semantics of every expression in the language of Table II is given as an SQL query. Formally, for any such expression $x$ in Table II, the function $\mathsf{SQL}(x)$ gives the SQL query that $x$ maps to. Below, we define SQL inductively from the simplest expressions to the most complex ones. For the purposes of this paper we give the definitions in pseudocode.

TABLE II
SYNTAX OF CANARY

| | | |
|---|---|---|
| <query> | : | <expr> \| <arg_expr> |
| <expr> | : | <req_expr> \| <sol_expr> |
| <req_expr> | : | requirement \| requirement where <condition> \| <aggregator> ( <req_expr> ) |
| <sol_expr> | : | solution ( <req_expr> ) \| solution ( <req_expr> ) where <condition> \| <aggregator> ( <sol_expr> ) |
| <arg_expr> | : | <arg_entity> ( <expr> ) \| <arg_entity> ( <expr> ) where <condition> |
| <arg_entity> | : | support \| rebuttal |
| <aggregator> | : | popular \| unpopular \| controversial \| discussed |

SQL(requirement). This gives the SQL query to return all the comments expressing requirements. We then traverse their trees (comment-reply structure of the discussions) to reduce their propagated scores. In the example in Listing 1 posScore is the sum of the score of positive interaction (supports or derived objects), posInter is the count of how many positive objects of interest are in the tree. Calculating the negative side of the tree is done the same way.

Listing 1. SQL(requirement)
```
select comment.*, sum(posScore),
    count(posInter),
    sum(negScore), count(negInter) from (
  select comment-reply-tree in (
    select * from comment
    join requirement
    on comment.id = requirement.idcomment  ) )
```

SQL(x) where $\phi$. Gives the SQL query to return all $x$ that satisfy the condition $\phi$. In essence, $\phi$ acts as a selection filter, as shown in Listing 2.

Listing 2. SQL(x) where <condition>
```
select * from SQL (x) where  φ
```

SQL(solution(x)). Since solutions must be related to a requirement to make sense, we must link them to the requirements they satisfy. We accomplish this by joining the solutions table and SQL(requirement), in this case represented as SQL(x). Listing 3 is the SQL pseudocode for the definition.

Listing 3. SQL(solution(x))
```
select comment.*, sum(posScore),
    count(posInter),
    sum(negScore), count(negInter) from (
  select comment-reply-tree in (
    select * from comment
    join solution
    on comment.id = solution.idcomment  )
) as solution
join SQL (x) as requirement
  on solution.idparent = requirement.idcomment
```

SQL(support(x)). Similarly, in order to find supporting arguments for an RE entity, we first traverse tree, then reduce the score, and then join it with the RE entity itself (in this case represented as SQL(x)), using a common foreign key.

SQL(rebuttal(x)). Finding rebuttal arguments for an RE entity is analogous: we traverse the tree, then reduce the score, and join it with the RE entity itself (represented as SQL(x)), using a common foreign key.

Canary supports aggregators such as *discussed*, *popular*, *unpopular*, and *controversial* to allow a selection of objects of interest based on aggregate metrics.

SQL(discussed(x)). We define a requirement to be discussed if the sum of the number of replies (support and rebuttals) and upvotes and downvotes is greater than some threshold. We find the number of supports by using the SQL count function and grouping by the foreign key we will use for the join with the entity. We compute the number of rebuttals in an analogous way. Then we join with SQL(x). The resulting relation has both counts of support and rebuttal comments as attributes, and we apply the threshold condition $\alpha$ so we get only those $x$ with values above threshold. Note that the value of $\alpha$ will be configured by the analyst. For the purposes of this paper, we set $\alpha = 10$.

Listing 4. SQL(discussed(x))

```
SQL (x) as object
where (object.rebs + object.sups) > 10
```

The sets of *popular*, *unpopular*, and *controversial* records are all subsets of *discussed*. They are all discussed entities, where either the positive sentiment dominates (popular), or the negative sentiment dominates (unpopular), or there is a balance between the two (controversial). We define positive sentiment as the sum of the number of support and upvotes and negative sentiment as the sum of the number of rebuttals and downvotes. In order to calculate them, we encapsulate SQL(discussed(x)) in a select statement and apply the appropriate selection filter to it.

SQL(popular(x)). Gives all discussed $x$ where the ratio of positive to negative sentiment is greater than $\beta$. Again, $\beta$ is configurable by the analyst. For this paper, we set it to 1.15, as shown in Listing 5.

Listing 5. SQL(popular(x))

```
select * from(
  SQL (discussed(x))
) as object
where ((object.supsScore * object.sups) /
  (object.rebsScore * object.rebs) ) > 1.15
```

SQL(unpopular(x)). Gives all discussed $x$ where the ratio of positive to negative sentiment is less than $\theta$. Again, $\theta$ is configurable by the analyst. For this paper, we set it to 0.85.

SQL(controversial(x)). Gives all discussed $x$ where the ratio of positive to negative sentiment lies between $\theta$ and $\beta$.

### D. Implementation of Canary Compiler

We implemented a compiler for Canary syntax in Java. We use the Eclipse XText (version 2.9) language definition and parsing library. The compiler takes queries written in Canary grammar and generates SQL queries following the definitions in Section III-C that can then be run on the aforementioned database. To accomplish this, the compiler essentially takes advantage of XText facilities. Given a Canary expression, XText creates a parse tree of a Canary expression based on the grammar and allows a recursive traversal of the tree, plugging in the SQL expressions that each node in the tree maps to.

## IV. Evaluation

Canary exploits a crowdsourcing approach to acquire annotations for users' comments. In this section, we evaluate the *feasibility* of this crowdsourcing approach. To be feasible, we conjecture that, the crowdsourcing approach must yield high-quality annotations, and be efficient. Accordingly, we seek to answer the following research questions.

$Q_1$. What is the *quality* of the crowd-acquired annotations?
$Q_2$. How *efficient* is the annotation process for the crowd?

### A. User Study

We conducted Canary's feasibility study employing Amazon Mechanical Turk (MTurk) users. As study units, we selected five online discussions from two social forums, each involving user discussions about Google Maps and related software applications. Table III summarizes these discussions (actual discussions are available in an online appendix [5]). Initially, we planned to include more discussions from Google Forums. However, we noticed that unlike Reddit discussions which involved a rich variety of user interactions, Google Forum discussions were much simpler in that they involved a few requirements and a huge number of supports (Table IV). Thus, we decided against adding more Google Forum discussions since that would not likely influence potential conclusions.

TABLE III
Summary of the online discussions we employed in our study

| Disc. | Comments | Words | Source | Discussion title |
|---|---|---|---|---|
| 1 | 141 | 6034 | Reddit | Feature Google Now should add: "Nearest x that is still open" |
| 2 | 184 | 13333 | Reddit | There is no way to properly save an address in Google Maps |
| 3 | 79 | 3975 | Reddit | Google Maps should use your average walking speed from Google Fit to calculate walking times |
| 4 | 261 | 8654 | Reddit | Google maps should have an "I need gas" feature... |
| 5 | 218 | 11310 | Google Forum | Make 'avoid tolls' option sticky otherwise directions are wrong |

A typical online discussion starts with a topic (e.g., a question) and several top-level threads fork from it. These discussions, in general, tend to be long. A lay user may require several hours to annotate one full discussion. For tasks crowdsourced to lay users, Cheng et al. [6] find that requiring users to perform smaller parts (*microtasks*) of a large task (*macrotask*) yields higher output quality, completion rate, and experience than requiring workers to perform the macrotask (our tasks are similar to the ones Cheng et al. study in that both require analytical and language understanding abilities). Accordingly, we decided to split the discussions in our study into smaller chunks of approximately equal length ($\mu = 1882$ and $\sigma = 229$ words). Each microtask in our study included the topic of a discussion followed by one or more threads about it. We decided not to split the top-level threads so as to preserve the context of interactions (we note, however, that some interactions may span top-level threads, but from our experience such instances are rare).

Splitting the discussions yielded 38 microtasks. We sought to acquire annotations from two users for each microtask so as to get a reliable estimate. Accordingly, we launched 76 HITs (human intensive tasks) on Amazon MTurk and collected annotations from 44 unique MTurk users. We restricted participants to be from majority native English speaking countries; UK, USA, Canada, Australia, and New Zealand. Overall, we rejected three HITs as incomplete. We paid USD 4 for each successful HIT. Our study was approved by the Ethics Board at Lancaster University and we received an informed consent from each participant. Complete questionnaires and ethics documents are available the online appendix [5].

As part of the study, the MTurk users were asked to (1) answer a pre-survey about demographics, (2) complete a main task, and (3) answer a post-survey about time, difficulty, and the understanding of instructions and the concepts.

In the main task, first, we asked users to read about the core concepts of requirements, solutions, supports and rebuttals from a document we provided. The document included multiple examples for each concept from online discussions. Next, we asked users to download a PDF file consisting of a discussion chunk and annotate the file via Adobe Acrobat Reader. We provided instructions to install the software, a tutorial on performing annotations, and an example PDF file with relevant annotations. Finally, we asked users to upload the annotated PDF file via an URL we provided.

We instructed users to select any comment or a part of a comment in the provided discussion and annotate it as one of the four entities: *requirement*, *solution*, *support*, or *rebuttal*. For simplicity, we asked users to annotation a piece of text with at most one entity. Further, for each annotation, we asked users to indicate their confidence in the annotation on a Likert scale of *very low*, *low*, *medium*, *high*, and *very high*.

In the post-survey, we asked participants to report the time they spent for the main task (i.e., time for reading and annotating the PDF file with the discussion, excluding the time for pre- and post-surveys). We also asked participants, the difficulty of the main task, and how well they understood the concepts of requirements, solutions, supports, and rebuttals, each on a scale of 1 (very low) to 5 (very high). Finally, we asked users to provide additional comments, if any.

## B. Ground Truth

A key challenge in evaluating the quality of annotations is establishing a standard for comparison. To establish the ground truth, the first two authors of the paper, acting as expert annotators, annotated each discussion in three rounds. In the first round, the two experts annotated the discussions independently without seeing each others' annotations. In the second round, they saw each others' annotations and updated their annotations, independently. In the third round, the experts discussed their annotations, resolved differences, and settled on one set of annotations as the *ground truth*.

Table IV summarises the experts' annotations. Note that the two experts were in complete agreement after the third round.

TABLE IV
A SUMMARY OF EXPERT ANNOTATIONS FOR EACH DISCUSSION

| Disc. | Requirement | Solution | Support | Rebuttal |
|---|---|---|---|---|
| 1 | 26 | 20 | 17 | 13 |
| 2 | 53 | 50 | 27 | 29 |
| 3 | 12 | 3 | 10 | 17 |
| 4 | 51 | 65 | 16 | 12 |
| 5 | 21 | 7 | 127 | 3 |

## C. Measures

After expert and MTurk users' annotations, we prepared a dataset for measuring quality as follows. First, for each discussion, we collected all pieces of text that had an annotation (from experts or users). For each such piece of text, we assigned an *expert-label* and a *user-label* as follows. The expert-agreed annotation (requirement, solution, support, or rebuttal) of a piece of text is its expert-label. If a piece of text in the list was not annotated by experts, we assigned *none* as its expert-label. Next, for each piece of text, if the two users' annotations for the text match (irrespective of confidence), we assigned the corresponding annotation as the *user-label* for the piece of text. Further, for a piece of text such that the two annotations do not match, but one annotation has a confidence of high or very high, and is higher than the confidence of the other annotation, we assigned the annotation with the highest confidence as the user-label for the text. We assigned none as the user-label for all remaining pieces of the text.

For each discussion, given the list of annotated text, and their expert- and user-labels, we measured the quality of user annotations via the following metrics.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}; \quad \text{Recall} = \frac{\text{TP}}{\text{TP + FN}};$$
$$F_1\text{-score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}};$$

where TP, FP, TN, and FN refer to true and false positives and negatives, respectively.

## D. Results

*1) Quality:* Table V shows the confusion matrix comparing the expert- and user-labels, aggregating counts across all five discussions. Table VI shows the mean and variance of the per-discussion precision, recall, and $F_1$ scores.

First, we observe that a vast majority of MTurk users' annotations are true positives (diagonal elements in the confusion matrix). Considering the complexity of the annotation task, we believe that the overall quality of user annotations is quite promising. Specifically, the precision for requirements annotations is very high. Second, we observe that the counts in the row corresponding to the *none* class-label are quite low

TABLE V
THE CONFUSION MATRIX COMPARING EXPERTS' AND MTURK USERS' ANNOTATIONS (COUNTS AGGREGATED FOR FIVE DISCUSSIONS)

| | | MTurk User Annotation | | | | |
|---|---|---|---|---|---|---|
| | | Req. | Sol. | Sup. | Reb. | None |
| Expert Annotation | Req. | 122 | 3 | 6 | 3 | 29 |
| | Sol. | 1 | 107 | 14 | 4 | 19 |
| | Sup. | 1 | 7 | 131 | 1 | 57 |
| | Reb. | 2 | 0 | 3 | 51 | 18 |
| | None | 3 | 9 | 7 | 8 | – |

TABLE VI
THE QUALITY OF MTURK USERS' ANNOTATIONS

| | Precision | | Recall | | $F_1$ Score | |
|---|---|---|---|---|---|---|
| | Mean | Var. | Mean | SD | Mean | SD |
| Requirement | 0.94 | 0.05 | 0.71 | 0.12 | 0.81 | 0.10 |
| Solution | 0.72 | 0.33 | 0.70 | 0.22 | 0.70 | 0.28 |
| Support | 0.76 | 0.19 | 0.68 | 0.19 | 0.67 | 0.10 |
| Rebuttal | 0.76 | 0.18 | 0.67 | 0.33 | 0.68 | 0.28 |

in the confusion matrix. This indicates that users are quite effective in distinguishing text containing requirements-related information from noisy comments (i.e., comments irrelevant for RE) that abundant in online discussions.

*2) Efficiency:* Figure 13 (top-most box plot) shows the distribution of the durations reported by the users. The mean amount of time participants spent on the main task is about 35 minutes. However, the variance in time spent is high. A few users, in the comments, indicated that they were using the Adobe Reader software for the first time and we believe that there may be other such users in our sample. We conjecture that it is for such users that the main task duration is high.

We had a few returning users in our dataset ($n = 10$). The bottom two box plots in Figure 13 compare the durations reported by these users for the first task and the second tasks. We find that users take significantly less time the second time ($p = 0.02$; measured via Wilcoxon's ranksum test, excluding outliers). This suggests that lay users can annotate online discussions in a time-efficient manner once they are familiar with the concepts and tools.

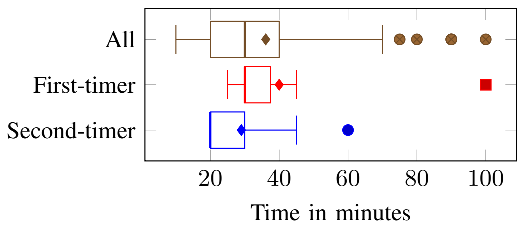


Fig. 13. Times spent by MTurk users for one annotation task

Figure 14 shows the distributions of clarity and difficulty ratings reported by MTurk users. A vast majority of users rated their understanding of instructions and the concepts involved as high or very high. Further, the difficulty ratings suggest that the annotation task is of moderate difficulty.

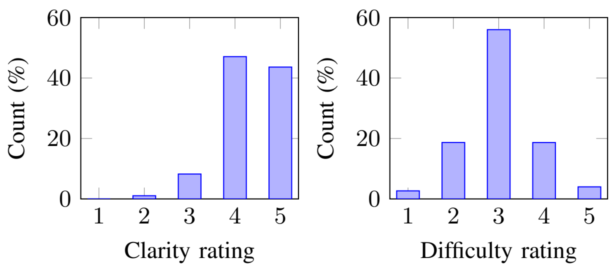


Fig. 14. Clarity and difficulty ratings provided by MTurk users

Finally, we manually analyzed 24 comments that users provided. We found 11 comments to be conveying a positive, 8 neutral, and 6 negative comments. The negative sentiment comments mainly indicated that some annotations can be ambiguous; one comment indicated that annotations in Adobe Reader are clunky. However, many of the positive comments indicated the task to be fun and interesting.

### E. Threats to Validity

We identify an internal threat to validity of our results. The authors of the paper who performed expert annotations were also part of designing the evaluation. Although the authors performed expert annotations systematically, in multiple rounds, there is a slight risk that the experts have subconsciously tried to second-guess how lay users are likely to annotate. Future studies can mitigate this risk by employing third-party experts. For this study we only targeted Google Maps as a target application for the discussions. We also only used Reddit and Google Forums as a source of discussion, with a focus on Reddit. Variations in the social media sources might affect the results as Canary is closely tied to the raw data input.

## V. RELATED WORK

*Crowdsourcing and user feedback.* Numerous services now support user feedback on applications. For example, UserVoice [2] attracts participation from a large community of users and has elicited thousands of bugs, problems, and suggestions for improvements. Likewise, user feedback on Google Play and Apple Appstore has been used toward gaining a better understanding of requirements for the next release [27].

User feedback and crowd-based RE have long been argued as essential for the RE process [26], [9]. Tools have been developed to enable crowdsourcing requirements in enterprise settings. StakeRare [15] is a methodology for identifying stakeholders and requirements using collaborative filtering based on social networks. Seyff et al. [32] show how general purpose social networking sites can support requirements elicitation, prioritization, and negotiation. Johann and Maalej [14] discuss giving users varying degrees of influence in RE by using different e-democracy strategies. Gamification is another approach for deeper inclusion of end-users in RE. iThink [7] and REfine [33] are platforms that apply gamification toward requirements elicitation and refinement. A recent study by Lombriser et al. [16] evaluated application of gamification to RE and found that gamification improves the quantity and quality of elicited requirements and has a positive effect on motivation and participation. Canary complements these works by leveraging richer content creation by the crowd via annotations and enabling sophisticated queries of the content.

Some work builds requirements-related annotations into a user platform [3], [20]. The alternative would be to obtain the annotations offline, either manually or with natural language processing support [22]. We note ongoing efforts to develop conceptual models of online user discussions [21].

Various other benefits of involving the crowd in the RE process have been studied in literature, such as creativity [23], [11], [35]. Stimulating the creativity of the crowd by, e.g., interacting with each others' ideas, has been found to measurably increase the quality of elicited requirements. Canary is built upon data generated by interaction in the form of argumentation between users in online forums. Murukannaiah et al. [24] report that such argumentation promotes the elicitation of better requirements.

*Querying requirements.* IBM DOORS [13] enables capturing users discussions on requirements; however, querying is limited to text searches. Canary queries are significantly more sophisticated and would allow for the elicitation of more pertinent information. Tools such as DOORS would benefit from Canary. TiQi by Huang et al.[43] allows the transformation of spoken natural language into structured SQL. It's designed to make traceability information easily accessible to its users, similar to what we're trying to achieve with Canary and information generated in online interaction. ReqIF [44] makes the exchange of formalized requirements between autonomous business partners possible. It also allows for the creation of custom relations between objects in the database and other queries based on attributes and pattern matching on strings.

*Argumentation* has long been advocated as a way of recording the rationale of requirements [29]. In recent work, Yu et al. [39] apply Toulmin's argumentation schema [36] to automated reasoning about security requirements. The reasoning is akin to running queries on argumentation bases. Versions of Canary targeted toward expert users could support richer argumentation and reasoning.

*Natural language processing (NLP)* has traditionally been seen as a promising tool for requirements analysis and there is resurgence of interest, recently [30], [1], [8], [19], [28]. Pagano and Maalej [27] evaluate online feedback as a source of information and find that, with proper classification (into categories such as feature requests, bug reports, and sentiments about them), online feedback can be used to construct better requirements. Maalej and Nabil [17] provide NLP techniques for such classification with a high degree of precision and recall. Canary's annotations bear some similarity to Maleej and Nabil's categories; however, the online discussions we analyze are more complex in structure and richer in content. Maalej et al. [18] in fact discuss the lack of systematic approaches to organize, summarize, and aggregate data from user communities. Our approach using human intelligence is complementary to machine intelligence techniques.

## VI. Discussion

In this paper, we have presented Canary; a tool-supported approach for querying requirements-related artifacts from user discussions. The centerpiece of the approach is a high-level query language in which requirement analysts can pose simple but useful queries to take advantage of the social features of online discussions. Our query language has a translation into SQL, which means that queries can be executed against discussions stored in relational databases. We implemented a compiler and demonstrated the results of a few Canary queries on a database of real discussions. Analysts and developers may use Canary to inform their reasoning when compiling the list of formal requirements.

To obtain the metadata for storage in database, we obtained requirements and argumentation-related annotations from Mechanical Turk users. We demonstrated the efficacy of our approach for annotations by providing a detailed empirical analysis of the quality of annotations. Although the results are promising, we observe a high variance in results in Table VI. This suggests that the quality may vary for discussions.

Canary annotations are simpler than some requirements models in the literature. In particular, we did not consider conflict, priority, positive and negative contributions, and assumptions [4], [37] and argumentation [36]. Including these concepts in Canary would require considering their meaning in the context of user discussions. For instance, priority could map to a suitable notion of popularity. The tradeoff of richer requirements models is more complex annotation. It is conceivable that there would be a proliferation of conceptual models with various levels of technical sophistication and associated query languages and tools.

An important future direction is to augment Canary with automated annotation techniques based on NLP. Argumentation mining [42] has recently been applied to social media [41], [40] and Canary may be able to exploit argumentation mining toward automating annotations. Another interesting application of NLP would be to use NLP as the underlying query processing engine. Such an engine might, for example, detect discussions about similar requirements in two or more distinct discussions and merge the interaction from both discussion to calculate the output of the query.

The overall methodology is currently labour intensive. We had to extract data from online forums, and reproduce it in a format suitable for annotation by MTurk users, and then load the annotated data into the database. We created several custom tools to help us with the tasks, e.g., for extracting data from Reddit using its published API and scripts to load annotated data into a database. A future direction will be to build an automated tool chain.

## References

[1] Vincenzo Ambriola and Vincenzo Gervasi. On the systematic analysis of natural language requirements with CIRCE. *Automated Software Engineering*, 13(1):107–167, 2006.

[2] Dejana Bajic and Kelly Lyons. Leveraging social media to gather user feedback for software development. In *Proc. of the 2nd International Workshop on Web 2.0 for Software Engineering*, pages 1–6. ACM, 2011.

[3] Travis D. Breaux and Florian Schaub. Scaling requirements extraction to the crowd: Experiments with privacy policies. In *Proc. of RE*, pages 163–172, 2014.

[4] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[5] Canary supplementary information. http://www.lancaster.ac.uk/staff/chopraak/canary/index.html Accessed: 2017-02-18.

[6] Justin Cheng, Jaime Teevan, Shamsi T. Iqbal, and Michael S. Bernstein. Break it down: A comparison of macro-and microtasks. In *Proc. of CHI*, pages 4061–4064, 2015. ACM.

[7] João Fernandes, Diogo Duarte, Claudia Ribeiro, Carla Farinha, João Madeiras Pereira, and Miguel Mira da Silva. iThink: A game-based approach towards improving collaboration and participation in requirement elicitation. *Procedia Computer Science*, 15:66–77, 2012.

[8] Vincenzo Gervasi and Didar Zowghi. Supporting traceability through affinity mining. In *Proc. of RE*, pages 143–152. IEEE, 2014.

[9] Eduard C Groen, Joerg Doerr, and Sebastian Adam. Towards crowd-based requirements engineering a research preview. In *Proc. of REFSQ*, pages 247–253. Springer, 2015.

[10] Charles B. Haley, Robin C. Laney, Jonathan D. Moffett, and Bashar Nuseibeh. Security requirements engineering: A framework for representation and analysis. *IEEE TSE*, 34(1):133–153. 2008.

[11] Jennifer Horkoff, Neil Maiden, and James Lockerbie. Creativity and goal modeling for software requirements engineering. In *Proc. ACM SIGCHI Conference on Creativity and Cognition*, pages 165–168. 2015.

[12] Mahmood Hosseini, Keith Phalp, Jacqui Taylor, and Raian Ali. The four pillars of crowdsourcing: A reference model. In *Proc. RCIS*, pages 1–12. IEEE, 2014.

[13] IBM. IBM Rational DOORS product overview. https://www.ibm.com/support/knowledgecenter/SSYQBZ_9.6.1/com.ibm.doors.requirements.doc/topics/c_welcome.html. Accessed: 2016-03-15.

[14] Timo Johann and Walid Maalej. Democratic mass participation of users in requirements engineering? In *Proc. of RE*, pages 256–261, 2015.

[15] Soo Ling Lim and Anthony Finkelstein. Stakerare: Using social networks and collaborative filtering for large-scale requirements elicitation. *IEEE TSE*, 38(3):707–735, 2012.

[16] Philipp Lombriser, Fabiano Dalpiaz, Garm Lucassen, and Sjaak Brinkkemper. Gamified requirements engineering: Model and experimentation. In *Proc. of REFSQ*, pages 171–187. Springer, 2016.

[17] Walid Maalej and Hadeer Nabil. Bug report, feature request, or simply praise? On automatically classifying app reviews. In *Proc. of RE*, pages 116–125, 2015.

[18] Walid Maalej, Maleknaz Nayebi, Timo Johann, and Guenther Ruhe. Toward data-driven requirements engineering. *IEEE Software*, 33(1):48–54, 2016.

[19] William Martin, Federica Sarro, Yue Jia, Yuanyuan Zhang, and Mark Harman. A survey of app store analysis for software engineering. *IEEE TSE*, 2016.

[20] Itzel Morales-Ramirez, Dimitra Papadimitriou, and Anna Perini. Crowd intent: Annotation of intentions hidden in online discussions. In *Proc. of 2nd IEEE/ACM International Workshop on CrowdSourcing in Software Engineering*, pages 24–29, 2015.

[21] Itzel Morales-Ramirez, Anna Perini, and Renata S. S. Guizzardi. An ontology of online user feedback in software engineering. *Applied Ontology*, 10(3-4):297–330, 2015.

[22] Itzel Morales-Ramirez, Matthieu Vergne, Mirko Morandini, Anna Perini, and Angelo Susi. Exploiting online discussions in collaborative distributed requirements engineering. In *Proc. of i* Workshop*, pages 7–12, 2015.

[23] Pradeep K. Murukannaiah, Nirav Ajmeri, and Munindar P. Singh. Acquiring creative requirements from the crowd: Understanding the influences of personality and creative potential in crowd RE. In *Proc. of RE*, pages 176–185, 2016.

[24] Pradeep K. Murukannaiah, Anup K. Kalia, Pankaj R. Telang, and Munindar P. Singh. Resolving goal conflicts via argumentation-based analysis of competing hypotheses. In *Proc. of RE*, pages 156–165, 2015.

[25] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: A roadmap. In *Proc. of FSE*, pages 35–46. ACM, 2000.

[26] Dennis Pagano and Bernd Brügge. User involvement in software evolution practice: A case study. In *Proc. of ICSE*, pages 953–962. IEEE, 2013.

[27] Dennis Pagano and Walid Maalej. User feedback in the appstore: An empirical study. In *Proc. of RE*, pages 125–134, 2013.

[28] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado Aaron Visaggio, Gerardo Canfora, and Harald C. Gall. Ardoc: app reviews development oriented classifier. In *Proc. of FSE*, pages 1023–1027, 2016.

[29] Balasubramaniam Ramesh and Vasant Dhar. Supporting systems development by capturing deliberations during requirements engineering. *IEEE TSE*, 18(6):498–510, June 1992.

[30] Pete Sawyer, Paul Rayson, and Ken Cosh. Shallow knowledge as an aid to deep understanding in early phase requirements engineering. *IEEE TSE*, 31(11):969–981, November 2005.

[31] Norbert Seyff, Florian Graf, and Neil Maiden. Using mobile RE tools to give end-users their own voice. In *Proc. of RE*, pages 37–46, 2010.

[32] Norbert Seyff, Irina Todoran, Kevin Caluser, Leif Singer, and Martin Glinz. Using popular social network sites to support requirements elicitation, prioritization and negotiation. *Journal of Internet Services and Applications*, 6(1):1–16, 2015.

[33] Remco Snijders, Fabiano Dalpiaz, Sjaak Brinkkemper, Mahmood Hosseini, Raian Ali, and Atilla Ozum. REfine: A gamified platform for participatory requirements engineering. In *Proceedings of 1st International Workshop on Crowd-Based Requirements Engineering*, pages 1–6. IEEE, 2015.

[34] Alistair Sutcliffe and Pete Sawyer. Requirements elicitation: Towards the unknown unknowns. In *Proc. of RE*, pages 92–104. IEEE, 2013.

[35] Richard Berntsson Svensson and Maryam Taghavianfar. Selecting creativity techniques for creative requirements: An evaluation of four techniques using creativity workshops. In *Proc. of RE*, pages 66–75. IEEE, 2015.

[36] Stephen E. Toulmin. *The Uses of Argument*. Cambridge University Press, Cambridge, UK, 1958.

[37] Axel van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, Chichester, UK, 2009.

[38] Jon Whittle, William Simm, Maria-Angela Ferrario, Katerina Frankova, Laurence Garton, Andrée Woodcock, Baseerit Nasa, Jane Binner, and Aom Ariyatum. VoiceYourView: Collecting real-time feedback on the design of public spaces. In *Proc. UbiCom*, pages 41–50, 2010.

[39] Yijun Yu, Virginia N.L. Franqueira, Thein Than Tun, Roel J. Wieringa, and Bashar Nuseibeh. Automated analysis of security requirements through risk-based argumentation. *Journal of Systems and Software*, 106:102–116, 2015.

[40] Filip Boltuzic and Jan Snajder. Back up your stance: Recognizing arguments in online discussions. In *Proc. of the First Workshop on Argumentation Mining*, 49–58, 2014.

[41] Joonsuk Park and Claire Cardie. Identifying appropriate support for propositions in online user comments. In *Proc. of the First Workshop on Argumentation Mining*, 29–38, 2014.

[42] Marie-Francine Moens and Erik Boiy and Raquel Mochales Palau and Chris Reed. Automatic detection of arguments in legal texts. In *Proc. of the 11th International Conference on Artificial Intelligence and Law*, 225–230, 2007.

[43] Piotr Pruski, Sugandha Lohar, Rundale Aquanette, Greg Ott, Sorawit Amornborvornwong, Alexander Rasin, and Jane Cleland-Huang. Tiqi: Towards natural language trace queries. In *Proc. of RE*,123–132, 2014.

[44] Christof Ebert and Michael Jastram. ReqIF: Seamless requirements interchange format between business partners. *IEEE Software*, 29(5):82–87, 2012.