Answering the Requirements Traceability Questions

Anonymous Author(s)

ABSTRACT

To understand requirements traceability in practice, we contribute in this paper an automated approach to identifying questions from requirements repositories and examining their answering status. Applying our approach to 345 open-source projects results in 20,622 questions, among which 53% and 15% are classified as successfully and unsuccessfully answered respectively. By building the requirements socio-technical graphs, we explore the impact of stakeholder-artifact relationships on traceability. The number of people, surprisingly, has little influence compared to other graph-theoretic measures like clustering coefficient. Based on the repository mining results, we formulate a novel set of hypotheses about traceability. A case study supports some hypotheses while offering new insights.

CCS CONCEPTS

• **Software and its engineering → Requirements analysis**; **Traceability**;

KEYWORDS

Requirements traceability, traceability questions, practitioner questions, repository mining, socio-technical graphs

ACM Reference Format:
Anonymous Author(s). 2018. Answering the Requirements Traceability Questions. In *Proceedings of International Conference on Software Engineering, Gothenburg, Sweden, May 27–June 3 (ICSE'18),* 12 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The *requirements traceability problem* refers to the inability to describe and follow the life of a requirement [28]. Despite many advances, requirements traceability remains a problem area for software engineering practitioners. In safety-critical industries, for example, traceability failures can lead regulators like the US Food and Drug Administration to cast doubt in product safety [46].

However, the acceptance of the requirements traceability problem is by no means universal in practice. For example, the majority of the project leaders interviewed by Blaauboer *et al.* [13] acknowledged the lack of awareness of requirements traceability, let alone instrumenting traceability in their software development projects. A rather extreme position was taken by the owner of a rapidly expanding IT company who understood the concept of traceability but viewed it as a "made up problem" [20].

ICSE'18, May 27–June 3, Gothenburg, Sweden

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The perceptions lie in not only the technical artifacts but also the social structures of software engineering. What are perceived to be requirements traceability problems, as studied by Gotel and Finkelstein [30], tend to arise when practitioners are unable to answer questions about the personnel who had been involved in the production and refinement of requirements. Although five example questions were defined and investigated in [30], many issues are still under-explored. What questions do practitioners actually ask regarding the life of a requirement? Whether and how are they answered during software development? Is the requirements traceability information less needed, or not needed at all, in certain cases? If so, which kind of requirements, and why? These illustrate the motivations behind our research.

Prior work has helped identify practitioner concerns about requirements traceability. The seminal work by Ramesh and Jarke [59] classified the low-end and high-end traceability usage based on the focus groups and interviews conducted in 26 software organizations with 136 participants. Bouillon *et al.* [16] compiled a list of 29 requirements traceability usage scenarios, such as change effort estimation and test coverage analysis. Their survey of 56 participants showed that about half of the scenarios were performed regularly in practice. While these studies revealed the tasks and activities that the participants wanted traceability to support, Lohar *et al.* [45] recently organized a conference event to collect natural language trace queries that practitioners would like to ask. As a result, over 300 queries were gathered from almost 50 participants [44].

In this paper, we build on and significantly extend previous research by proposing a repository mining approach to not only identify the requirements traceability questions asked by software stakeholders, but also analyze whether the questions are answered successfully or if they are answered at all. We apply our approach to 345 open-source projects where 117,590 requirements are communicated, refined, realized, and evolved. From the online data, we mined 20,622 questions posted by 2,528 different people querying the requirements life. To our surprise, only a very few questions were about compliance verification or other commonly cited scenarios [16]. New traceability practices emerged, e.g., managing cross-project requirements interactions in the software ecosystem.

Our results further show that, during software development, 53% of the traceability questions were addressed successfully, 15% unsuccessfully, and 32% were not answered. Examining the answering status suggests the role of socio-technical relationships play in describing and following the life of a requirement. We therefore take a graph-theoretic perspective to explore the measures of a requirements-centric network, mainly its order and clumpiness [19]. The socio-technical understandings enable us to formulate a novel set of hypotheses about the requirements traceability problem. We test the hypotheses in a case study where traceability questions' asking and answering are investigated in the real-life context.

The main contributions of this paper are three-fold: (1) we develop automated methods to mine requirements traceability questions and classify their answering status, (2) we evaluate and apply

our methods with open-source repositories, and (3) we derive new hypotheses from a socio-technical perspective and test them in a case study to complement the post-hoc repository mining analyses. In what follows, we review related work in Section 2 and present our repository mining study in Section 3. The socio-technical networks of requirements traceability are built and analyzed in Section 4. Section 5 describes our case study, and Section 6 concludes.

2 BACKGROUND AND RELATED WORK

2.1 Practitioner Questions about Requirements Traceability

The current state of traceability practice spans a spectrum, from fast-paced, agile-like, and business-oriented applications on one end, to slower-paced, carefully planned, and safety-critical projects on the other [20]. Comprehensive knowledge exists in the safety-critical domains where traceability is governed by standards. Our work builds on the views of requirements traceability from practitioners working on different software projects, including those in which safety, security, and other dependability concerns are critical.

One of today's most commonly adopted definitions of requirements traceability came from Gotel and Finkelstein's empirical studies involving more than 100 practitioners, with whom 5 focus groups, a 2-stage questionnaire, 2 interview sessions, and various observations were carried out [28]. They pointed out that the weak application of traceability in practice was mainly due to the process and stakeholders of requirements engineering.

Process-wise, it is not sufficient to begin tracking a requirement's life only after that requirement is included in the requirements specification (RS). In addition to the post-RS traceability, pre-RS tracing shall be instrumented to make clear those aspects of a requirement's life prior to its inclusion in the RS. Altogether, the pre-RS and post-RS dimensions delineate *requirements traceability*, which refers to "the ability to describe and follow the life of a requirement, in both a forwards and backwards direction" [28]. We note that, in many open-source and agile projects including the ones that we studied, the pre-RS and post-RS traceability becomes more integrated because there is seldom a clear point in the process to define the RS. Rather, the requirements are continuously clarified and refined among the software stakeholders [5, 25, 40].

The stakeholders are foundational to traceability because people are the ultimate baseline whenever requirements need to be re-examined or re-worked. In fact, when practitioners know the personnel underlying the requirements, traceability tends not to be perceived as a problem [30]. Contribution structures are proposed to model stakeholders' social roles, e.g., the principal, author, and documentor of the RS [29]. Applying contribution structures in a communications company with around 25 employees demonstrated how the network of people made it possible to answer 5 questions regarding requirements involvement, responsibility, ramifications, change notifications, and working arrangements [30]. Our work expands contribution structures by fusing technical artifacts in the stakeholder network.

While the 5 questions were asked by the researchers in a post-hoc manner [30], practitioners' opinions were surveyed and interviewed that suggested a wide range of traceability practices. An earlier work by Ramesh and Jarke distinguished two groups [59]. The low-end used traceability to satisfy the mandate from the sponsors or for compliance with standards, and the high-end exploited traceability to manage decisions and rationales throughout the project lifecycle. A more recent survey by Bouillon *et al.* [16] found that 42% of the usage scenarios commonly cited in the literature were applied regularly in practice. On the top of the list were tracking the progress of a requirement's implementation, pre-RS traceability to regulatory and other source of a requirement, and compliance demonstration. The results of our study show differences in the traceability uses and uncover new questions asked by practitioners.

Similar to our work, Lohar and colleagues were interested in what requirements traceability questions that practitioners would express in their own words [44, 45]. Prompted with 9 usage scenarios, the participants issued many trace queries, e.g., "Is any of the open fault logs related to an exception case?", "Which use cases have more than 5 requirements associated with them?", etc. Different from our work, the gathered queries were anchored by the EasyClinic and Isolette systems [45, 56, 57]. In contrast, we study practitioners' trace queries in their own development projects.

2.2 Socio-Technical Theories in Software Engineering

Contemporary approaches to requirements traceability are either artifact-based (e.g., trace retrieval [15]) or driven by social roles (e.g., contribution structures in the RS production [29]). We review two strands of research: a socio-technical theory of coordination (STTC) [35] and the Codebook framework [11], because they not only integrate explicitly the social and technical aspects of software engineering, but provide unifying powers to address practitioner questions in a principled way. We also present some preliminaries to situate our research in the context of graph mining [19].

To understand "who should work with whom" in globally distributed software development, Herbsleb and colleagues [35] developed a STTC in which the dependencies among engineering decisions are seen as defining a constraint satisfaction problem that people must organize to solve. The theory draws from the key observation that the number of people was a powerful predictor of coordination problems (e.g., delay in development [37]), and quantifies people's working relationships via a dependency network of work items. In particular, two undirected graphs are integrated through matrix multiplication [17, 18]: developer-file task assignment and file-file logical dependency. The resulting matrix theorizes "who should work with whom", which in turn serves as a unifying basis for solving a class of coordination problems, including expertise finding [51, 61], cross-site communication [36], and pull request acceptance in social coding environments [67]. While this STTC can address some of the traceability questions (e.g., ramifications [30]), other questions, such as "Which programmers are more error prone in their code according to the test results?" [44], need the relationships between people and artifacts.

Such relationships are exploited in the Codebook framework [11]. By using a single data structure (graph of people and artifacts mined from software repositories) and a single algorithm (regular language reachability), Codebook could handle all the inter-team coordination problems identified in a survey responded by 110 Microsoft

employees. Because Codebook is general, practitioner questions need to be cast into regular expressions by some domain expert. For example, to trace "Which program manager wrote the specification for that code?", the regular expression "Code MentionedBy WordDocument AuthoredBy Person" shall be prepared and answered through the paths returned by the reachability algorithm. We adopt Codebook's underlying data structure, but instead of regular language reachability, we look into the measures of the people-artifact graph itself for questions about the life of a requirement.

The patterns observed frequently in synthetic but realistic graphs include: power-law degree distributions, small diameters making pairs of nodes reachable, and community effects resulted from a clumpy structure of the graph where each node is closer to the other nodes within the community (cluster) than to nodes outside it [19]. Chakrabarti and Faloutsos further pointed out that power laws are absent in most social networks, in part due to the fact that power-law patterns can be observed reliably only in large datasets [19]. For this reason, we pay attention to: (1) the clumpiness (clustering coefficient), and (2) the order (number of nodes) of the requirements socio-technical graph. The latter subsumes the number of people, a critical factor in collaborative software engineering [35].

3 UNVEILING REQUIREMENTS TRACEABILITY QUESTIONS

The objective of our repository mining study is to develop automated ways for identifying requirements traceability questions and analyzing how they are answered. The key construct is "requirements traceability question", which we define as a query that a project stakeholder issues *in situ* wanting to know a requirement's life. Similarly, it is the stakeholders' social interactions recorded during software development that are mined in order for us to classify each question's answering status: unanswered, answered successfully, or answered unsuccessfully.

3.1 Project Selection

Building on Rempel and Mäder's recent traceability analysis of 24 open-source projects with industrial relevance [60], we chose 345 projects in our study due to the public availability of development artifacts and traceability data.[1] All of our subject projects are from the Apache software foundation [7] or the JBoss family [38]. To be included in our study, an open-source project must be under active development and have already evolved over some stable releases to ensure project maturity.

Among the 345 projects, we further selected 6 to serve as benchmarks for the evaluation of our automated methods. Table 1 lists the characteristics and statistics of the 6 projects. For each benchmark project, we manually created the answer sets[2] of requirements traceability questions and their answering status. It is against these answer sets that the accuracy of our methods is measured. The resulting accuracies represent our methods' estimated performance levels because devising the answer sets for all the 345 projects is impractical.

[1] We share our collected data and analysis results in [6] for validation and replication purposes.
[2] Our answer sets can also be found in [6].

The 6 benchmark projects tackle problems in different domains with implementations written in different programming languages. For each project, we extract the data from its initial release to its latest stable release. The information is shown in Table 1. Our guiding principle of choosing the benchmarks is to balance different dimensions so as to enhance our methods' validity among the pool of the 345 projects. To that end, five metrics are collected for all the 345 projects to inform our benchmark selections:

- Average release cycle indicating the project pace;
- Total number of requirements from the initial to the latest releases implying the project size;
- Average number of comments communicated on each requirement showing the intensity of social interactions;
- Average number of stakeholders measuring the size of the group contributing to the requirement; and
- Average number of code files associated with the requirement suggesting the implementation scope.

Shapiro-Wilk test rejects distribution normality for all the above 5 measures, making it less appropriate to directly use means or standard deviations. We therefore divide the 345 projects into quartiles along each of the 5 measures. Figure 1 presents the radar chart, in which the minimum, first quartile, median, third quartile, and maximum values of each measure are specified. The figure shows that each benchmark project crosses different quartiles on different measures. Meanwhile, no quartile of Figure 1 is left uncovered by our benchmark projects. Such a combination of balance and coverage helps us determine the use of *six* projects as benchmarks, and further increases our confidence about the generalizability of our automated methods when assessed against these 6 projects, despite that our 5 gauging measures are by no means exhaustive and may not be completely independent either.

3.2 Identifying Requirements Traceability Questions

In order to automatically find requirements traceability questions, it is crucial to recognize where such questions are likely to be recorded. The literature suggests that issue trackers like JIRA are essential for open-source projects to manage requirements [5, 25, 33, 40, 60]. Although the requirements of an open-source project can originate from emails, how-to guides, and other socially lightweight sources [62], the to-be-implemented requirements "eventually end up as feature requests in an issue tracking system" [33]. In addition, a project's SCM (software configuration management) system such as Git or Subversion contains commit logs with references to artifact identifiers of the issue tracker.

Similar to [60], issue trackers and SCM systems are central to our traceability study. Common to all our 345 subject systems is the use of JIRA and Git for lifecycle management. Extending beyond [60], we realize two additional repositories: forums and IRC channels, in which questions are likely to be posed. Google groups are representative forums allowing developers to seek help and advice, discuss new ideas, or share user-related stories [38]. The intended forum usage is not for requirements traceability questions but for 'how to' questions related to software setup. Our observations are consistent with the intended usage in that specific forum questions include: "How to deploy two instances of the same rar file on JBoss

Table 1: Six benchmark projects used to evaluate the mining of requirements traceability questions and their answers.

Project	Characteristics					Statistics				
	domain	written in	initial release	latest release	latest version	# of days per release	# of req.s	# of comments per req.	# of stakehold -ers per req.	# of code files per req.
AIRFLOW	workflow execution	Python	Oct 16 2014	Feb 2 2017	1.8.0	30.42	629	2.16	2.11	11.76
ANY23	RDF data extraction	Java	Jul 16 2012	Feb 26 2017	2.0	139.52	182	3.54	2.14	20.33
DASH-BUILDER	data reports	Java HTML	Aug 27 2014	Apr 14 2016	6.4.0. Final	42.73	85	1.49	1.71	26.13
DROOLS	business rules	Java	Nov 13 2012	Jul 17 2017	7.1.0 Final	29.36	486	1.53	1.78	31.26
IMMU-TANT	complexity reduction	Clojure	Mar 14 2012	Jun 23 2017	2.1.9	51.27	404	1.15	1.33	3.17
JBTM	business process	Java C++	Dec 5 2005	Jul 14 2017	5.6.3. Final	21.23	1575	1.81	1.63	32.37

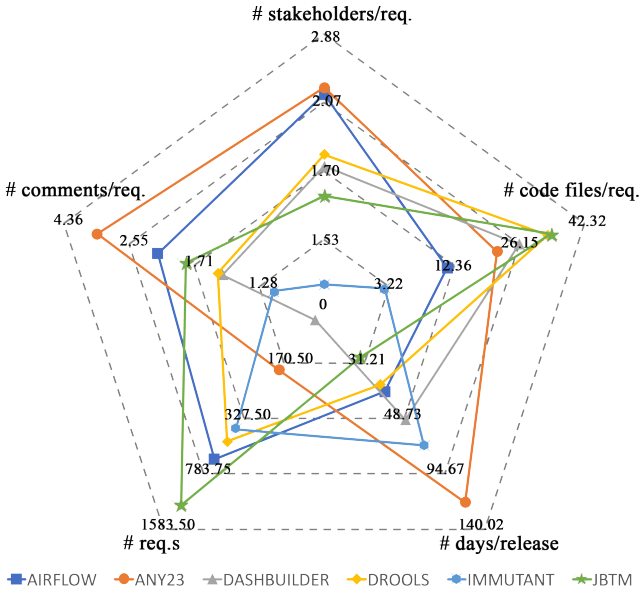


Figure 1: Balancing the selection of benchmark projects.

Input: project's requirements repositories (e.g., JIRA)
Output: set of questions extracted from the repositories

Main Procedure
1. recognize all the requirements (R) in the repositories
2. **For** each requirement $r \in R$
3. $C_r \leftarrow$ all the comments of r
4. remove duplicates from C_r
5. remove irrelevant information from C_r
6. **For** each comment $c \in C_r$
7. **If** c contains a question tagged by the NLP tool
8. $Q \leftarrow Q \cup \{c\}$
9. $Q \leftarrow$ **Co-reference**(C_r, Q)
10. **Return** Q

Co-reference(C_r, Q)
11. **For** each comment $c \in C_r$
12. **If** $\exists$ a sentence $s \in c$ **such that** (s has no verb) **and** (s refers to an entity $e \in C_r - \{c\}$)
13. $Q \leftarrow Q \cup \{c\}$

Figure 2: Finding questions in requirements repositories.

6.4 EAP?" [55], "How to configure WildFly 10 to send JMS messages from a Stateless Session EJB?" [65], etc.

Among our subject systems, two main usages are intended for IRC: real-time communications as in WildFly [70], and focused topic discussions as in AeroGear [1]. As a result, requirements traceability questions may arise, and so will many other types of questions. Nevertheless, our observations support the primary role that issue trackers play in managing requirements [33] and their traceability information [34]. For example, the IRC minutes containing some traceability questions [2] are linked as part of an issue's comment [68].

The reason that we pay much attention to the repositories is because, with their suitable selections, "identifying requirements traceability questions" can be formulated into "identifying questions from requirements repositories". Issue trackers (specifically JIRA) are core to our current operations, and SCM systems (Git) are also considered. We exclude developer forums (Google groups) and plan to integrate IRC channels in the future. Figure 2 outlines our algorithm for question identification by leveraging state-of-the-art NLP (natural language processing) tools. In our work, the Stanford CoreNLP toolkit [47] is used.

To recognize requirements (Line 1 of Figure 2), we rely on the issue type defined in each of the 345 projects. Figure 3 shows the common requirements types and the number of projects using them. Although "task" and "feature request" are widely adopted, their decompositions and refinements like "sub-task" and "new

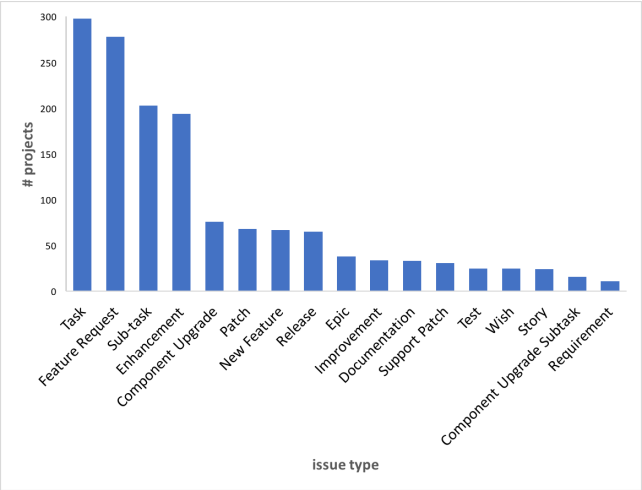


Figure 3: Using issue types to recognize project requirements. Only the types adopted by more than 10 projects are shown here; the entire list of 29 types is available at [6].

Table 2: Evaluating the identification requirements traceability questions.

Project	|Answer Set|	$\|Q\|$	Recall	Precision	F_1
AIRFLOW	145	155	0.92	0.86	0.89
ANY23	98	96	0.77	0.78	0.77
DASHBUILDER	18	15	0.78	0.93	0.85
DROOLS	96	109	0.96	0.84	0.89
IMMUTANT	41	48	0.95	0.81	0.87
JBTM	394	465	0.99	0.84	0.91

feature" exist. The requirements of "epic", "test", and "story" reflect the project's agile nature.

For every recognized requirement, our algorithm then collects its comments (C_r) and pre-processes C_r in two steps (Lines 3–5 of Figure 2). The first step discards duplicates which commonly occur when a comment reply includes the content that has already been posted before. Our current duplication removal is based on "{quote}" and "bq." (block quotation) marked in JIRA's comment formatting technique. The second step filters out three kinds of irrelevant information: xml, url, and source code. The question mark appeared in xml tags or url links confuses Stanford CoreNLP's parsing, and so do the code snippets.

Two levels of question identification are performed: a single comment (Lines 6–8 of Figure 2), and the set of all comments. We handle the latter by a co-reference procedure to detect the sentence without any verb yet with reference to some entity belonging to other comments of the requirement. For example, to follow up on the implementation of a DROOLS requirement [27], one simply asked: "Any status on this." Statements like this are so brief that the essential verb evoking semantic frames in every known language [26] is missing. When such statements refer to an entity (e.g., a noun) from the other comment in the context, our co-reference procedure flags them as questions. The returned results (Q in Figure 2) are at the comment level, aiming to balance self-containedness off too little (a sentence) or too much (a set of comments) information.

We evaluate the results Q by comparing them with the manually created answer sets of the 6 benchmark projects. These comparisons are summarized in Table 2. For each benchmark, two researchers built their answer sets individually and reached a substantial degree of agreement: average Cohen's kappa=0.67 on requirements traceability questions over 6 projects. The discrepancies were resolved in a joint meeting between the researchers. Although researchers have analyzed practitioner questions mined from software repositories (e.g., [4, 10, 66]), none was concerned about distinguishing questions from non-questions. Recent work by Liu *et al.* [43] investigated supervised learning methods for automatically detecting questions in microblogs. Their experiment with 8,465 Sina microblogs (1,278 questions) showed that the combination of lexical, syntax, and contextual features achieved the best performance: Recall=0.73, Precision=0.63, and F_1=0.68. Our results presented in Table 2 are comparable to this performance level, though our algorithm is unsupervised and mines questions from requirements repositories.

Analyzing the errors (false positives) and misses (false negatives) from the 6 benchmarks offers ways for future improvement. For example, our algorithm misses questions like: "It would be a great help if a workaround is provided to have default ascending sort of category" asking about the implementation completeness of the "data sorting" requirement [52]. This style of traceability questioning may also be signaled with "show me", "list all", or "I'd like to see" [44, 45], making the incorporation of certain indicator terms a promising extension of our algorithm.

Examining the results from all the 345 projects provides some quantitative views and also challenges some conventional perceptions. Quantitatively speaking, 93% (19,173 out of 20,622) of the questions were identified at the single comment level (Lines 6–8 of Figure 2), showing the applicability as well as the power of state-of-the-art NLP tools. On the other hand, 7% of the questions were uncovered by considering relevant contextual information via our co-reference procedure, illustrating a novel application of the general-purpose NLP tools in mining requirements repositories. Contrary to what people stated how they used requirements traceability in surveys like [16], our mining results show that only a small handful of questions that practitioners actually asked (e.g., [22]) were about compliance verification or in the broad category of verification and validation (V&V). One reason might be that the V&V questions tend to be asked more often in *after-the-fact* tracing activities [32] but less so *during* project development.

In line with the previous survey results [16], our study shows that tracking the progress of a requirement's implementation was among the most commonly asked questions. What is worth noting here is the tracking scope often spans across the project where the requirement resides. For example, by asking the patch status of DROOLS's requirement aimed to "support overloading of accumulate functions" [53], Michael Neale showed interests in not only the broader issues of data customization, but the DROOLS project itself. A couple of months later, he requested a feature in another project (TorqueBox) to "have smooth DROOLS integration" where one could write rules directly against AR (Rails ActiveRecord) models and have facts punted into the engine in an

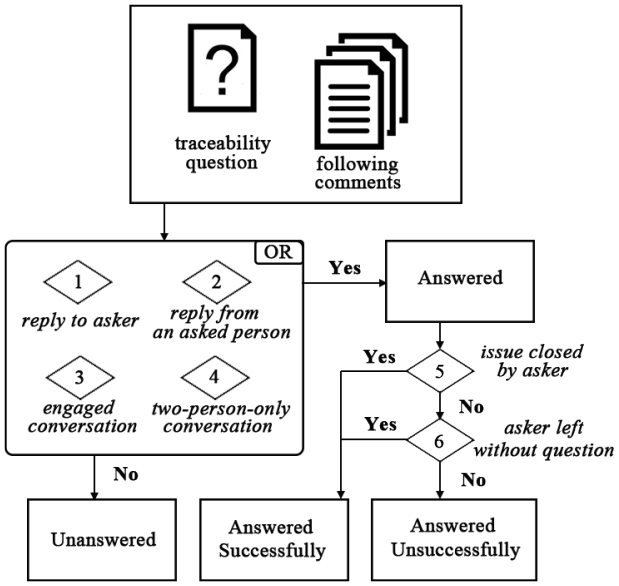


Figure 4: Classifying the requirements traceability question's answering status.

appropriate form [54]. The cross-project linkage illustrates that the upstream/downstream dependencies between projects in a software ecosystem [14] could be rooted in requirements interactions, and these requirements-level dependencies could be captured and understood by the traceability questions.

3.3 Classifying Answering Status

For each identified requirements traceability question, its answering status is automatically classified by our procedure shown in Figure 4. The input consists of the question q that the person p_q asked about the requirement r at time t_q, as well as C_f representing all the comments about r that were posted after t_q. We call C_f the set of following comments with respect to q, and use P_f to denote the set of authors of C_f. Our automated method of Figure 4 then checks if any of the four conditions are met:

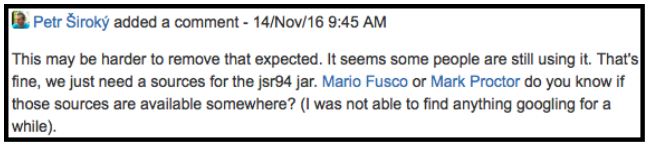
(1) reply to asker: $\exists\ c_f \in C_f$ such that c_f mentioned a_q;
(2) reply from an asked person: if q mentioned anybody from the set P_m, then $P_m \cap P_f \neq \phi$;
(3) engaged conversation: $\exists\ c_x, c_y \in C_f$, c_x was created by p_x at t_x, c_y was created by a_q at t_y such that (c_y mentioned p_x) and ($t_x < t_y$); or
(4) two-person-only conversation: $|\, C_f - \{a_q\}\,| = 1$.

If a question failed to meet *all* the four conditions, then we consider it to be unanswered; otherwise, the relevant information from C_f suggests that there existed attempts to answer q. To distinguish whether the answering was successful or unsuccessful, two more steps in Figure 4 are executed:

(5) issue closed by asker: $\exists\ t_c > t_q$ such that a_q marked r as 'done' or 'fixed' at t_c; and
(6) asker left without question: if $c_l \in \{q\} \cup C_f$ and c_l was a_q's last comment in C_f, then $c_l \notin Q$.

Compared to the disjunction of conditions (1)-(4), we impose an ordering of (5) followed by (6). Our rationale is that closing the issue, along with people's question-answer interactions, has a strong indication of the requirement being fulfilled. If the closing comes from the asker, then his or her question tends to be resolved from a practical point of view, making it unnecessary to check other conditions. Finally, a question is classified as answered unsuccessfully if the asker left the discussions about a not-yet-resolved requirement with some question, i.e., his or her last comment is part of Q identified by the algorithm of Figure 2.

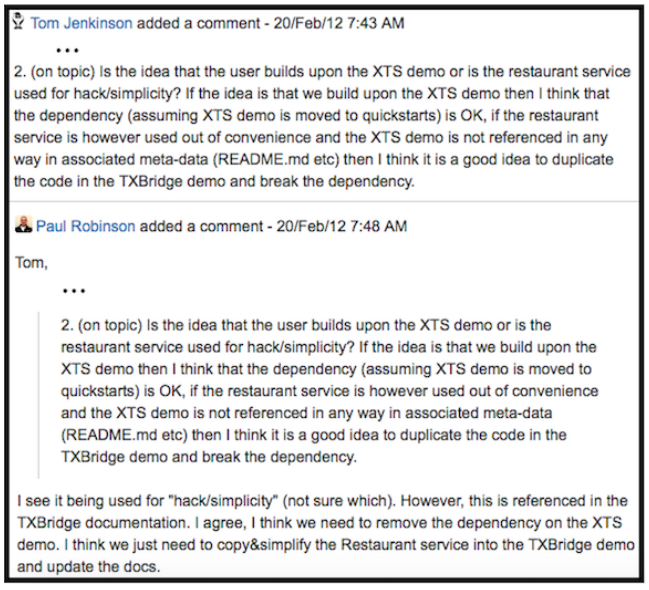
We illustrate the classification procedure of Figure 4 with three example questions, one from each category. The following snapshot shows an unanswered question. Petr Široký proposed to drop the support for JSR94 integration in DROOLS and assigned himself to work on this requirements task [64]. He wanted to find the jsr94 jar and directed his question specifically to two people.


Petr Široký added a comment - 14/Nov/16 9:45 AM

This may be harder to remove that expected. It seems some people are still using it. That's fine, we just need a sources for the jsr94 jar. Mario Fusco or Mark Proctor do you know if those sources are available somewhere? (I was not able to find anything googling for a while).

Unfortunately, Petr's question failed all the conditions (1)-(4) of Figure 4. In particular, neither of the asked people replied. Despite that Petr later closed this issue, he commented [64]: "This is not going to happen anytime soon (if ever), so I am closing the JIRA." Our method of Figure 4 is therefore able to correctly classify Petr's question as unanswered.

Our method classifies the JBTM question below asked by Tom Jenkinson [39] as successfully answered. Tom wanted to know the correct way to interpret the porting requirement so as to decide if the built artifacts' dependency should be broken. When checking this question and the following comments, conditions (1) and (4) of Figure 4 are met. Furthermore, Tom closed this issue based on Paul's response suggesting the removal of the dependency [39].


Tom Jenkinson added a comment - 20/Feb/12 7:43 AM

...

2. (on topic) Is the idea that the user builds upon the XTS demo or is the restaurant service used for hack/simplicity? If the idea is that we build upon the XTS demo then I think that the dependency (assuming XTS demo is moved to quickstarts) is OK, if the restaurant service is however used out of convenience and the XTS demo is not referenced in any way in associated meta-data (README.md etc) then I think it is a good idea to duplicate the code in the TXBridge demo and break the dependency.

Paul Robinson added a comment - 20/Feb/12 7:48 AM

Tom,

...

> 2. (on topic) Is the idea that the user builds upon the XTS demo or is the restaurant service used for hack/simplicity? If the idea is that we build upon the XTS demo then I think that the dependency (assuming XTS demo is moved to quickstarts) is OK, if the restaurant service is however used out of convenience and the XTS demo is not referenced in any way in associated meta-data (README.md etc) then I think it is a good idea to duplicate the code in the TXBridge demo and break the dependency.

I see it being used for "hack/simplicity" (not sure which). However, this is referenced in the TXBridge documentation. I agree, I think we need to remove the dependency on the XTS demo. I think we just need to copy&simplify the Restaurant service into the TXBridge demo and update the docs.

Table 3: Evaluating the classification of requirements traceability questions' answering status: "AS" means "answer set" constructed manually, "CR" means "classification results" obtained automatically by our method (cf. Figure 4), "R" means "recall", and "P" means "precision".

Project	Unanswered					Successfully Answered					Unsuccessfully Answered				
	|AS|	|CR|	R	P	F_1	|AS|	|CR|	R	P	F_1	|AS|	|CR|	R	P	F_1
AIRFLOW	32	38	0.75	0.63	0.69	67	52	0.70	0.90	0.79	46	55	0.80	0.67	0.73
ANY23	15	13	0.60	0.69	0.64	59	63	0.93	0.87	0.90	20	22	0.80	0.73	0.76
DASHBUILDER	4	5	1.00	0.80	0.89	10	10	0.90	0.90	0.90	4	3	0.75	1.00	0.86
DROOLS	29	28	0.83	0.86	0.84	46	45	0.83	0.84	0.84	21	23	0.81	0.74	0.77
IMMUTANT	17	15	0.82	0.93	0.88	21	22	0.90	0.86	0.89	3	4	1.00	0.75	0.86
JBTM	79	104	0.66	0.50	0.57	148	146	0.74	0.75	0.74	168	144	0.69	0.81	0.74

The following question from DROOLS [31] is among the questions classified into the unsuccessfully answered category. The two-person-only conversation style satisfies condition (4), indicating some answering attempt was made. Continuing in Figure 4, however, both conditions (5) and (6) are dissatisfied. In fact, the answerer below mentioned, "I can not guarantee that this is enough", showing a lack of confidence in the answer. As a result, the asker left this discussion thread with a limited ability to describe and follow the life of a component upgrade requirement, especially the backward compatibility issues when the rule flow is called in PackageBuilders [31].

Our evaluation of the classification results uses the 6 benchmark projects. The evaluation results are presented in Table 3. We built the answering-status answer sets in the same way as the traceability-question answer sets: individually first followed by joint consolidations (Cohen's κ=0.61). The additional information here is that, for each successfully answered question in our answer sets, we manually located its answer which we judged to be the correct answer. This ensures the quality of our answer sets which we make available at [6].

To assess the results shown in Table 3, we compare them with the state-of-the-art in computational linguistics [3, 63] and software development Q&A mining [9, 58]. Shrestha and McKeown [63] identified question-answer pairs in email conversations by using lexical and structural features, and showed the accuracy level of their method at Recall=0.62, Precision=0.70, and F_1=0.66. Agichtein *et al.* [3] concerned about finding high-quality answers according to human-labeled judgments, and reported the high-quality content classification achieved Recall=0.91, Precision=0.73, and F_1=0.81 on an Yahoo! Answers experiment. Note that these levels of accuracy should be compared only to the "successfully answered" classification but not to the other two categories of Table 3.

The work by Rahman and Roy [58] can shed light on our "unanswered" question classification. They introduced a prediction model

Figure 5: Classifying the 20,622 questions.

to identify unresolved questions at Stack Overflow, and showed in an experiment of 8,057 questions the prediction accuracy with Recall=0.76, Precision=0.79, and F_1=0.77. However, "unresolved" in [58] refers to a question's final status, whereas "unanswered" in our work focuses on the answering attempts. According to Arora *et al.* [9], 12% of some 8 million Stack Overflow questions received no answers. Moreover, their proposed approach achieved Recall=0.64, Precision=0.80, and F_1=0.71 in identifying the questions that would receive negative scores due to community's voting down. These negatively scored questions might be "unsuccessfully answered", but the downvotes indicate which questions and answers are least useful to the community [9].

In summary, our method presented in Figure 4 is novel in that it leverages the information from the answering process and the final status to classify whether the requirements traceability questions are answered successfully or if they are answered at all. Although no prior work has tackled the exact same issues, the assessments shown in Table 3 suggest our method's accuracy is comparable to the contemporary approaches of Q&A mining. Figure 5 presents the results of applying our method to the 345 projects' 20,622 traceability questions: 6,485 (31.45%) were unanswered, 11,013 (53.40%) were answered successfully, and 3,124 (15.15%) were answered unsuccessfully.

3.4 Threats to Validity

We discuss in this sub-section some of the most important factors that must be considered when interpreting the results of our repository mining study. The *construct validity* can be affected by our formulation of "identifying requirements traceability questions" into "identifying questions from requirements repositories". While such a formulation allowed for great automation, traceability questions might also be recorded in other repositories (e.g., mailing lists)

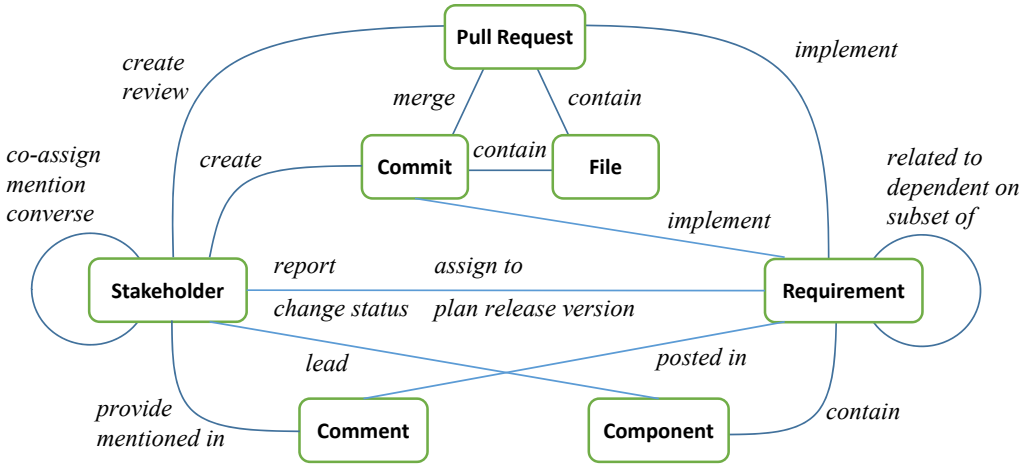


Figure 6: Node and edge types of RSTG (requirements socio-technical graph).

or asked through other channels (e.g., design meetings). Inside the requirements repositories like JIRA, not all questions were about requirements traceability. The false positives commonly encountered in our benchmark projects were the requests targeted at *all* the project issues, as opposed to only the requirements ones. A couple of examples from ANY23 [48, 49] could serve as illustrations: "Can you do a silent retarget (no notifications sent to mailing list) of all Jira issues to 2.0 to reflect the version bump?" and "Is there a comprehensive list of the changes on Any23?"

One of the *internal validity* threats relates to our sequential investigation of traceability questions and their answering status. Our construction of the answer sets for the 6 benchmark projects thus took this ordering into consideration. For example, although 15 questions were automatically identified in DASHBUILDER (cf. Table 2), we analyzed the 18 questions in the answer sets presented in Table 3: 4 unanswered, 10 successfully answered, and 4 unsuccessfully answered questions. Our rationale is that, once the answer set of requirements traceability questions is known, it makes little sense to worry about the answering status of the false positives, i.e., the automatically identified questions that were not requirements traceability questions. For the 6 benchmark projects, our method shown in Figure 4 was therefore applied to all and only the correct traceability questions. In situations where the answer sets are not known (e.g., our 345 subject systems excluding the 6 benchmarks), caution must be taken in interpreting the accuracy levels of our automations as a whole rather than separately.

Because of our effort in choosing the benchmark projects (cf. Section 3.1), *external validity* has been explored within the 345 subject systems but remains to be assessed beyond them. In particular, proprietary and/or safety-critical software projects are likely to exhibit different characteristics in pace, size, requirements communication, etc. Finally, in terms of *reliability*, we conduct all the analyses on open-source projects using procedures, algorithms, and measures completely described in this section. Moreover, our artifacts and supplementary materials are available at [6]. We therefore believe it is possible to independently replicate our results.

4 REQUIREMENTS SOCIO-TECHNICAL GRAPH (RSTG)

When analyzing the answering status of the questions, we realize the impact of the people-artifact relationships on the ability to trace a requirement's life. For example, Petr's question mentioned in Section 3.3 was very specific in terms of both the to-be-found artifact (jsr94 jar) and the people that he felt would be most helpful (Mario Fusco or Mark Proctor) [64]. Yet, the question was unanswered.

Checking the contribution structures [30] of this DROOLS requirement [64] leads to the social network with only one node, i.e., Petr who reported/created and was assigned to work on the requirement. By expanding the contribution structures to also include artifacts in the Codebook fashion [11], the number of nodes increases, e.g., Petr's comment containing the traceability question is added as an artifact node. Even in this socio-technical graph, the number of people nodes is rather small, with Petr, Mario, and Mark being involved. The inclusion of Mario and Mark is based only on the "mention" relationship of Petr and his comment. Our manual inspection of these three people in DROOLS suggests that they have relatively weaker connections among themselves, which in turn indicates that they may belong to different community-clusters.

Therefore, our objective in this section is to gain quantitatively insights into the socio-technical impacts on requirements traceability. Informed by the literature on graph mining [19] and our own observations, we focus on three measures of the *requirements socio-technical graph* (RSTG): number of nodes, number of people nodes, and clustering coefficient. We define the RSTG on top of the Codebook framework [11] with a special focus on requirements, stakeholders, and their relationships.

Figure 6 shows the types of nodes and edges considered in our current RSTG construction. Although most relationships in Figure 6 have a natural direction (e.g., stakeholder reports requirement, and not the other way around), some are symmetric (e.g., requirements relate to each other). Rather than considering a symmetric relationship in both ways, we build the RSTG as an undirected graph. Because all the relationships of Figure 6 are directly identifiable from the requirements repositories (namely, JIRA and Git in our

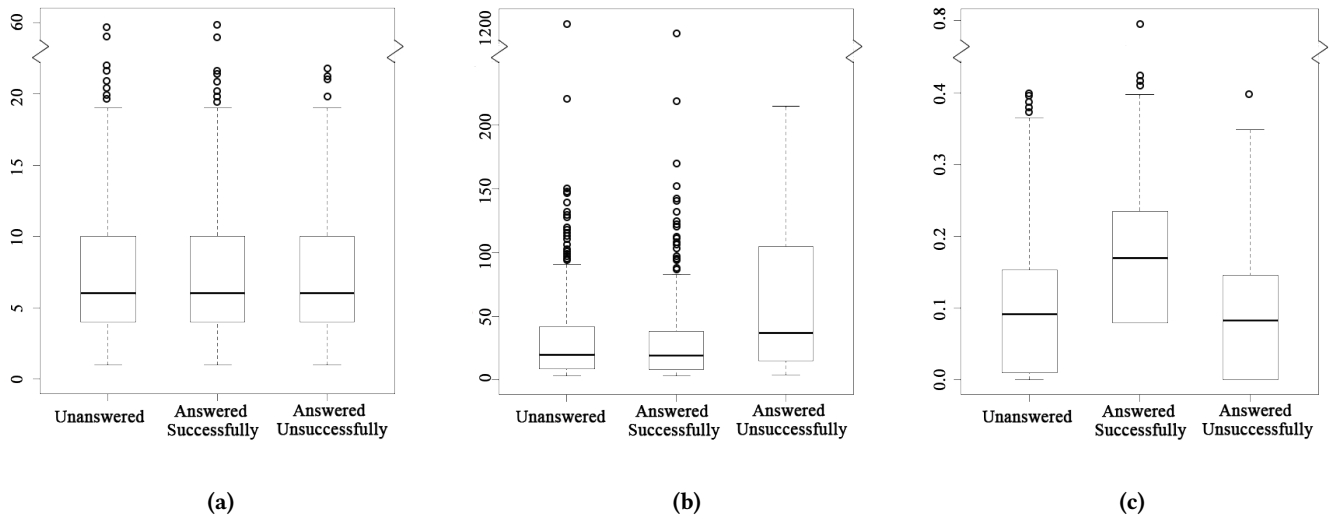


Figure 7: Graph-theoretic measures: (a) Number of people nodes, (b) Number of total nodes, and (c) Clustering coefficient.

study), the RSTG edges are unweighted. However, there may exist multiple kinds of relationships between a pair of nodes, e.g., two stakeholders can hold "co-assign", "mention", and/or "converse" relationships, and a stakeholder can "provide" (post) a comment and/or be "mentioned in" a comment. It is therefore the presence of edges, irrespective of their types, that determines the association strength of each pair of nodes in the RSTG.

To analyze the RSTG(r, q) specific to a given requirement r and its traceability question q, we adapt spreading activation [21, 42, 50] by selecting r and q to be the starting nodes. Spreading activation computes weights for nodes in two steps: pulses and termination checks. During pulses, we use normalized edge weights to transitively compute new weights for different nodes in the RSTG, allowing additional nodes to be activated. To prevent "flooding" of the spreading activation, we use the same termination check as [50], i.e., setting the values in the threshold function to terminate after 8 iterations. Finally, it is important to note that only the nodes and the edges existed *before* t_q are considered for the spreading activation of RSTG(r, q), where t_q is the time that q was asked.

Figure 7 explores the three measures of RSTG(r, q) for all the 20,622 questions that we studied, grouped by the questions' answering status. The results can be summarized from the viewpoint of each measure. The number of people nodes is not significantly different (one-way ANOVA: p=0.12). The number of total nodes is significantly greater in unsuccessfully answered category than the other two categories (p<0.05 for both independent t-tests). The clustering coefficient of successfully answered questions is significantly higher than the other questions (p<0.01 for both independent t-tests). These graph-theoretic results enable us to formulate a set of new hypotheses.

- **Hypothesis 1 (H1):** The number of people has a marginal influence on whether or how well the requirements traceability questions will be addressed.

  Although the number of people is shown to be a powerful predictor of delay and other coordination problems [35], it is not the case for the traceability problem. We could analogize this seemingly surprising result to Brooks's law. Here, for requirements traceability, more people cannot solve late questions, indicating the importance of *when* to ask, to answer, and to expect the answer. Another explanation of **H1** might be that it is not how many people, but *who* they are (e.g., core versus non-core developers [67, 69]), that determines how the requirements would be traced.

- **Hypothesis 2 (H2):** The more artifacts involved, the more likely the requirements traceability questions are addressed unsuccessfully.

  Taking the conventional artifact-oriented view, it seems obvious that traceability is more difficult when more dots were thrown out there to be connected. A deeper insight could be that most software artifacts (especially source code [23]) are inherently social, produced and consumed by overlapping yet different people. The provider-user gaps were long recognized in traceability [28]; recent work highlighted the new challenges in cross-disciplinary settings [71]. What is thought-provoking about **H2** to us is that less artifacts could lead to more successful traceability, and could also make people not handle traceability at all.

- **Hypothesis 3 (H3):** Requirements traceability tends to be addressed more successfully inside a community.

  Community effects have been found in many real-world graphs [19]. Software developers' social networks are no exception [12]. The latent community structures are reliable even if some software stakeholders are "hidden" [41] or some software artifacts are "secret" [8]. The challenge is to identify the core relationships, and not just the core stakeholder roles [30], of the community. These relationships, norms, and community values are not only project-specific [24] but also ecosystem-specific [14].

5 CASE STUDY

To complement the post-hoc nature of our repository mining study, we carried out a case study to investigate the traceability questions' asking and answering in the real-life context. Our partner company develops safety-critical software for controlling high-power devices and other embedded systems. Its division of software R&D consists of around 150 people, distributed globally in 4 sites: 2 in Europe, 1 in the US, and 1 in India. Different sites have their own expertise, e.g., safety, performance, simulation, and hardware-software co-design, etc. Nevertheless, collaborations including requirements tracing across sites are indispensable.

We collected the data for about a month from a selected software project's issue tracking system where the requirements are managed in an agile fashion, e.g., functional requirements are expressed as features, and epics help define customer needs as well as validation plans. One of the important relationships that the project members informed us to consider was the trainer-trainee relationship, and for that, we were provided with the relevant information such as the company's directory. By adapting the types of nodes and edges in Figure 6, we were able to construct the RSTG in our case study. Figure 8 shows RSTG(req1, comment85) where user85 asked a traceability question about req1 and this question appeared in comment85. Meanwhile, RSTG(req1, comment26) is also shown as part of Figure 8 where comment26 was raised earlier than comment85. In general, the spreading activation result of a later question is not a super graph of that of an earlier question. Figure 8 is therefore a special case where RSTG(req1, comment85) and RSTG(req1, comment26) can be viewed together.

We pay special attention to the community effect of RSTG(req1, comment85) and RSTG(req1, comment26). In the former, user291 interacted comment85 by posting comment77 and contributing a pull request (pr27) that contained file18. According to **H2** stated in Section 4, the increasing number of artifacts would make it harder to address req1's comment85 successfully. Our further analysis showed that user291 appeared in a different community from user85, thus relating to **H3**. We would predict that user291's responses likely leave comment85 not addressed properly, and our interview with user85 confirmed that. The main motivation for user291 to want to help was due to the collocation relationship. It turned out that, compared to the trainer-trainee relationship, being located at the same site was less important for the project that we studied. Our prediction about RSTG(req1, comment85) was correct, confirmed by the project members.

Following a similar reasoning, we predicted that comment26 would be addressed successfully by user31, who responded to the comment directly. However, this prediction was incorrect. In fact, the high clustering coefficient of RSTG(req1, comment26) related to **H3**, and the number of artifacts involved in RSTG(req1, comment26) was low, relating to **H2**. Yet, what we believed "successfully answered" turned out to be "unanswered", but meanwhile, the unanswered comment26 caused no practical concern to the asker (i.e., user85). What was confirmed was the community effect of user31 and user85 based on the latent structure of our RSTG.

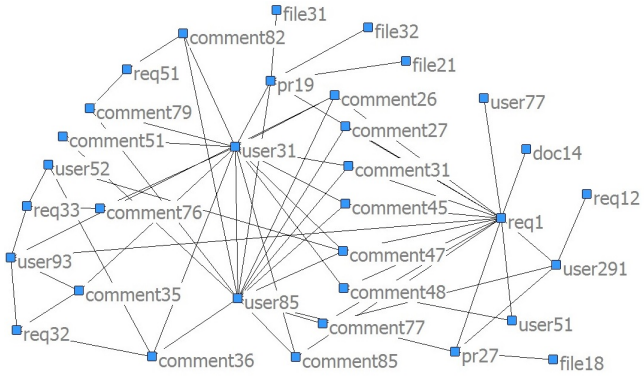


Figure 8: RSTG(req1, comment85) and RSTG(req1, comment26) in an integrated view.

6 CONCLUSIONS

Understanding practitioner questions helps keep research grounded and relevant. Prior work surveyed and interviewed participants about their traceability practices. What they stated could mismatch to some extent with what they actually did. In this paper, we present an automated approach to mining requirements repositories so as to identify questions and to classify whether and how well the questions are answered. Based on the quantitative results, we further contributed a socio-technical graph to exploit the stakeholder-requirements relationships and to explore the graph-theoretic measures commonly observed in realistic graphs.

Our future work includes incorporating indicator terms to better identify requirements traceability questions, refining the procedure for answering-status classification, and carrying out more empirical studies to test the formulated hypotheses.

REFERENCES

[1] AeroGear Community. https://aerogear.org/community. Last accessed: Aug 2017.
[2] AeroGear iOS Meeting Minutes. http://transcripts.jboss.org/meeting/irc.freenode.org/aerogear/2014/aerogear.2014-08-22-12.00.txt. Last accessed: Aug 2017.
[3] E. Agichtein, C. Castillo, D. Donato, A. Gionis, and G. Mishne. Finding high-quality content in social media. In *International Conference on Web Search and Web Data Mining (WSDM)*, pages 183–194, Stanford, CA, USA, February 2008.
[4] M. Ahasanuzzaman, M. Asaduzzaman, C. K. Roy, and K. A. Schneider. Mining duplicate questions in Stack Overflow. In *International Conference on Mining Software Repositories (MSR)*, pages 402–412, Austin, TX, USA, May 2016.
[5] T. A. Alspaugh and W. Scacchi. Ongoing software development without classical requirements. In *International Requirements Engineering Conference (RE)*, pages 165–174, Rio de Janeiro, Brazil, July 2013.
[6] Anonymous Authors. Artifacts of "answering the requirements traceability questions". https://goo.gl/BDaXvK. Last accessed: Aug 2017.
[7] Apache Software Foundation. http://www.apache.org. Last accessed: Aug 2017.
[8] J. Aranda and G. Venolia. The secret life of bugs: going past the errors and omissions in software repositories. In *International Conference on Software Engineering (ICSE)*, pages 298–308, Vancouver, Canada, May 2009.
[9] P. Arora, D. Ganguly, and G. J. F. Jones. The good, the bad and their kins: identifying questions with negative scores in StackOverflow. In *International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 1232–1239, Paris, France, August 2015.
[10] A. Barua, S. W. Thomas, and A. E. Hassan. What are developers talking about? an analysis of topics and trends in Stack Overflow. *Empirical Software Engineering*, 19(3):619–654, June 2014.
[11] A. Begel, Y. P. Khoo, and T. Zimmermann. Codebook: discovering and exploiting relationships in software repositories. In *International Conference on Software Engineering (ICSE)*, pages 125–134, Cape Town, South Africa, May 2010.

[12] C. Bird, D. S. Pattison, R. M. D'Souza, V. Filkov, and P. T. Devanbu. Latent social structure in open source projects. In *International Symposium on Foundations of Software Engineering (FSE)*, pages 24–35, Atlanta, GA, USA, November 2008.

[13] F. Blaauboer, K. Sikkel, and M. N. Aydin. Deciding to adopt requirements traceability in practice. In *International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 294–308, Trondheim, Norway, June 2007.

[14] C. Bogart, C. Kästner, J. D. Herbsleb, and F. Thung. How to break an API: cost negotiation and community values in three software ecosystems. In *International Symposium on Foundations of Software Engineering (FSE)*, pages 109–120, Seattle, WA, USA, November 2016.

[15] M. Borg, P. Runeson, and A. Ardö. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering*, 19(6):1565–1616, December 2014.

[16] E. Bouillon, P. Mäder, and I. Philippow. A survey on usage scenarios for requirements traceability in practice. In *International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 158–173, Essen, Germany, April 2013.

[17] M. Cataldo and J. D. Herbsleb. Coordination breakdowns and their impact on development productivity and software failures. *IEEE Transactions on Software Engineering*, 39(3):343–360, March 2013.

[18] M. Cataldo, J. D. Herbsleb, and K. M. Carley. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 2–11, Kaiserslautern, Germany, October 2016.

[19] D. Chakrabarti and C. Faloutsos. Graph mining: laws, generators, and algorithms. *ACM Computing Surveys*, 38(1): Article 2, March 2006.

[20] J. Cleland-Huang, O. Gotel, J. H. Hayes, P. Mäder, and A. Zisman. Software traceability: trends and future directions. In *International Conference on the Future of Software Engineering (FOSE)*, pages 55–69, Hyderabad, India, May-June 2014.

[21] A. M. Collins and E. F. Loftus. A spreading-activation theory of semantic processing. *Psychological Review*, 82(6):407–428, 1975.

[22] T. Crawley. https://issues.jboss.org/browse/IMMUTANT-219. Last accessed: Aug 2017.

[23] C. de Souza, J. Froehlich, and P. Dourish. Seeking the source: software source code as a social and technical artifact. In *International Conference on Supporting Group Work (GROUP)*, pages 197–206, Sanibel Island, FL, USA, November 2005.

[24] R. Dömges and K. Pohl. Adapting tracability environments to project-specific needs. *Communications of the ACM*, 41(12):54–62, December 1998.

[25] N. A. Ernst and G. C. Murphy. Case studies in just-in-time requirements analysis. In *International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 25–32, Chicago, IL, USA, September 2012.

[26] C. J. Fillmore. The case for case. In E. Bach and R. Harms, editors, *Universals in Linguistic Theory*, pages 1–88. Holt, Rinehart, and Winston, 1968.

[27] J. Goldsmith. https://issues.jboss.org/browse/DROOLS-1008. Last accessed: Aug 2017.

[28] O. Gotel and A. Finkelstein. An analysis of the requirements traceability problem. In *International Conference on Requirements Engineering (ICRE)*, pages 94–101, Colorado Springs, CO, USA, April 1994.

[29] O. Gotel and A. Finkelstein. Contribution structures. In *International Symposium on Requirements Engineering (RE)*, pages 100–107, York, UK, March 1995.

[30] O. Gotel and A. Finkelstein. Extended requirements traceability: results of an industrial case study. In *International Symposium on Requirements Engineering (RE)*, pages 169–178, Annapolis, MD, USA, January 1997.

[31] P. Hariharan and D. Sottara. https://issues.jboss.org/browse/DROOLS-147. Last accessed: Aug 2017.

[32] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram. Advancing candidate link generation for requirements tracing: the study of methods. *IEEE Transactions on Software Engineering*, 32(1):4–19, January 2006.

[33] P. Heck and A. Zaidman. An analysis of requirements evolution in open source projects: recommendations for issue trackers. In *International Workshop on Principles of Software Evolution (IWPSE)*, pages 43–52, Saint Petersburg, Russia, August 2013.

[34] P. Heck and A. Zaidman. Horizontal traceability for just-in-time requirements: the case for open source feature requests. *Journal of Software: Evolution and Process*, 26(12):1280–1296, December 2014.

[35] J. D. Herbsleb. Building a socio-technical theory of coordination: why and how. In *International Symposium on Foundations of Software Engineering (FSE)*, pages 2–10, Seattle, WA, USA, November 2016.

[36] J. D. Herbsleb, D. L. Atkins, D. G. Boyer, M. Handel, and T. A. Finholt. Introducing instant messaging and chat in the workplace. In *Conference on Human Factors in Computing Systems (CHI)*, pages 171–178, Minneapolis, MN, USA, April 2002.

[37] J. D. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering*, 29(6):481–494, June 2003.

[38] JBoss Family of Lightweight Cloud-Friendly Enterprise-Grade Products. http://www.jboss.org. Last accessed: Aug 2017.

[39] T. Jenkinson and P. Robinson. https://issues.jboss.org/browse/JBTM-1057. Last accessed: Aug 2017.

[40] E. Knauss, D. Damian, J. Cleland-Huang, and R. Helms. Patterns of continuous requirements clarification. *Requirements Engineering*, 20(4):383–403, November 2015.

[41] I. Kwan and D. Damian. The hidden experts in software-engineering communication. In *International Conference on Software Engineering (ICSE)*, pages 800–803, Honolulu, HI, USA, May 2011.

[42] J. Lawrance, C. Bogart, M. M. Burnett, R. K. E. Bellamy, K. Rector, and S. D. Fleming. How programmers debug, revisited: an information foraging theory perspective. *IEEE Transactions on Software Engineering*, 39(2):197–215, February 2013.

[43] X. Liu, R. Xie, C. Lin, and L. Cao. Question microblog identification and answer recommendation. *Multimedia Systems*, 22(4):487–496, July 2016.

[44] S. Lohar. Supporting natural language queries across the requirements engineering process. In *International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ) Doctoral Symposium*, Gothenburg, Sweden, March 2016.

[45] S. Lohar, J. Cleland-Huang, A. Rasin, and P. Mäder. Live study proposal: collecting natural language trace queries. In *International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ) Workshops*, pages 207–210, Essen, Germany, March 2015.

[46] P. Mäder, P. L. Jones, Y. Zhang, and J. Cleland-Huang. Strategic traceability for safety-critical projects. *IEEE Software*, 30(3):58–66, May/June 2013.

[47] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *Annual Meeting of the Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, Baltimore, MD, USA, June 2014.

[48] L. J. McGibbney. https://issues.apache.org/jira/browse/ANY23-36. Last accessed: Aug 2017.

[49] L. J. McGibbney and P. Ansell. https://issues.apache.org/jira/browse/ANY23-276. Last accessed: Aug 2017.

[50] C. McMillan, D. Poshyvanyk, M. Grechanik, Q. Xie, and C. Fu. Portfolio: searching for relevant functions and their usages in millions of lines of code. *ACM Transactions on Software Engineering and Methodology*, 22(4): Article 37, October 2013.

[51] A. Mockus and J. D. Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *International Conference on Software Engineering (ICSE)*, pages 503–512, Orlando, FL, USA, May 2002.

[52] M. Narang. https://issues.jboss.org/browse/DASHBUILDE-232. Last accessed: Aug 2017.

[53] M. Neale. https://issues.jboss.org/browse/DROOLS-315. Last accessed: Aug 2017.

[54] M. Neale. https://issues.jboss.org/browse/TORQUE-76. Last accessed: Aug 2017.

[55] pbudhran. https://developer.jboss.org/thread/275745. Last accessed: Aug 2017.

[56] P. Pruski, S. Lohar, R. Aquanette, G. Ott, S. Amornborvornwong, A. Rasin, and J. Cleland-Huang. TiQi: towards natural language trace queries. In *International Requirements Engineering Conference (RE)*, pages 123–132, Karlskrona, Sweden, August 2014.

[57] P. Pruski, S. Lohar, W. Goss, A. Rasin, and J. Cleland-Huang. TiQi: answering unstructured natural language trace queries. *Requirements Engineering*, 20(3):215–232, September 2015.

[58] M. M. Rahman and C. K. Roy. An insight into the unresolved questions at Stack Overflow. In *International Working Conference on Mining Software Repositories (MSR)*, pages 426–429, Florence, Italy, May 2015.

[59] B. Ramesh and M. Jarke. Toward reference models of requirements traceability. *IEEE Transactions on Software Engineering*, 27(1):58–93, January 2001.

[60] P. Rempel and P. Mäder. Preventing defects: the impact of requirements traceability completeness on software quality. *IEEE Transactions on Software Engineering*, 43(8):777–797, August 2017.

[61] A. Sarma, L. Maccherone, P. Wagstrom, and J. D. Herbsleb. Tesseract: interactive visual exploration of socio-technical relationships in software development. In *International Conference on Software Engineering (ICSE)*, pages 23–33, Vancouver, Canada, May 2009.

[62] W. Scacchi. Understanding the requirements for developing open source software systems. *IET Software*, 149(1):24–39, February 2002.

[63] L. Shrestha and K. McKeown. Detection of question-answer pairs in email conversations. In *International Conference on Computational Linguistics (COLING)*, Geneva, Switzerland, August 2004.

[64] P. Široký. https://issues.jboss.org/browse/DROOLS-1345. Last accessed: Aug 2017.

[65] R. Soika. https://developer.jboss.org/thread/275829. Last accessed: Aug 2017.

[66] C. Treude, O. Barzilay, and M.-A. D. Storey. How do programmers ask and answer questions on the web? In *International Conference on Software Engineering (ICSE)*, pages 804–807, Honolulu, HI, USA, May 2011.

[67] J. Tsay, L. Dabbish, and J. D. Herbsleb. Influence of social and technical factors for evaluating contribution in GitHub. In *International Conference on Software Engineering (ICSE)*, pages 356–366, Hyderabad, India, May-June 2014.

[68] C. Vasilakis. https://issues.jboss.org/browse/AGIOS-185. Last accessed: Aug 2017.
[69] B. Vasilescu, Y. Yu, H. Wang, P. T. Devanbu, and V. Filkov. Quality and productivity outcomes relating to continuous integration in GitHub. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 805–816, Bergamo, Italy, August-September 2015.
[70] WildFly Community. http://wildfly.org/gethelp. Last accessed: Aug 2017.
[71] R. Wohlrab, J.-P. Steghöfer, E. Knauss, S. Maro, and A. Anjorin. Collaborative traceability management: challenges and opportunities. In *International Requirements Engineering Conference (RE)*, pages 216–225, Beijing, China, September 2016.