Collecting and Analyzing Requirements Related Software Project Queries

Sugandha Malviya[*], Jane Cleland-Huang[**], and Alexander Rasin[*]

[*]DePaul University, College of Computing and Digital Media, Chicago, IL
[**]University of Notre Dame, South Bend, Indiana
[1]*slohar@depaul.edu, JaneClelandHuang@nd.edu, arasin@cdm.depaul.edu*

Abstract

Project data is often underutilized due to lack of accessibility and difficulties users have in constructing the non-trivial SQL queries that are often needed to support realistic Software Engineering questions. In our prior work we therefore presented TiQi – a natural language interface which allows users to issue trace queries in their own words. We initially collected sample queries from a limited number of industrial stakeholders, and utilized these to evaluate our proof of concept solution. As the work has matured, we have collected and analyzed a far more extensive set of trace queries from which we derive additional requirements for TiQi and construct a more challenging evaluation context. This paper not only describes the collected queries and their derived requirements, but discusses the need for iteratively building realistic benchmarks when engaging in research design projects.

1 TiQi: A Quick Overview

All software and systems engineering projects accumulate large amounts of artifacts in the form of requirements, design artifacts, code, test cases, project scheduling, fault logs, and other such data. This collection of data represents a valuable resource for managing various aspects of a software project; however, in practice it is often underutilized primarily because of the difficulties in accessing and retrieving it from heterogeneous data stores and then using it to formulate useful queries. To alleviate this problem, we have developed a natural language query interface, named TiQi for use in Software Engineering projects [1, 2].

TiQi[1, 2] prompts the user for a NL query by displaying a Traceability Information Model (TIM), which models artifact types, their attributes, and semantically typed links. TiQi utilizes an underlying domain model which includes *project-specific* terms and *question* terms. *Project-specific* terms are extracted from the raw data of the project. They include artifact names such as *hazard* or *test case*, attribute names such as *priority* or *created by*, and also terms extracted from the actual contents of the data. These terms are used to map the initial NL query to specific artifacts. *Question* terms form the 'glue' which holds the pieces of the query together. For example terms such as *show me*, *list all*, or *I'd like to see* are all synonyms for the SQL construct *SELECT*. Similarly, terms such as *associated with*, *related to*, or *with* are synonyms for various forms of *JOIN*.

In order to transform natural language queries to SQL, TiQi first parses and tokenizes the query, and then uses a series of heuristics and disambiguation techniques to map tokens onto artifacts and their attributes. Finally, it generates an executable SQL statement.

2 Datasets as a Core Benchmarking Mechanism

As with all Software and Requirements Engineering tasks, the reported success of an algorithmic solution is significantly influenced by the dataset against which it is evaluated. TiQi has three distinct types of data. The *query set* represents the set of natural language queries, the *project data* represents the software artifacts (i.e. requirements, design, code etc) against which the queries will be executed, and the *domain model* represents the knowledge TiQi currently possesses of synonyms, definitions, relationships between concepts in the domain, mapping rules, and disambiguation techniques.

Unsurprisingly, there is a strong interplay between all three datasets. First, the accuracy of query transformations tends to increase as the domain model is extended to capture more concepts and to encompass more sophisticated analytical and reasoning ability. Second, TiQi generally achieves higher accuracy scores when evaluated against simpler datasets of queries and smaller/simpler project datasets. Unsurprisingly, it tends to achieve lower accuracy scores when more complex queries and larger project datasets are introduced. From an empirical Software Engineering perspective - this introduces the interesting question of "**what constitutes a fair dataset at any particular phase of the research project?**"

2.1 Initial Query and Project Datasets

In our prior work we were initially concerned with delivering a *proof of concept* solution to demonstrate that TiQi was plausible [1]. The query set was created by inviting 10 traceability experts to generate queries based upon Traceability Information Models that we provided. Project data was extracted from two existing projects, Easy-Clinic and Isolette. However, in both cases the quantity and diversity of the data were quite limited. The focus of the work was on designing heuristics, algorithms, and mechanisms to transform trace queries to SQL – rather than on immediately delivering an industry-strength solution.

For a subsequent journal paper [2] we significantly expanded TiQi's domain knowledge and algorithmic capabilities, and extended the query set and project data against which the queries were tested. As a result, accuracy scores decreased despite the improvements made to TiQi. The decrease was attributed to an increase in the size and complexity of the project data and increased complexity of the query set. It is evident that we need to create a stable framework from which queries can be generated and evaluated for different project datasets.

2.2 Query Set Collection

To achieve this goal we conducted a live study at the Conference on Requirements Engineering for Software Quality in which we collected query samples from almost 50 different participants [3]. In the first phase of the study we provided a Traceability Information Model and presented various software engineering scenarios. Participants were asked to generate appropriate NL queries. This activity resulted in over 300 collected queries and exposed participants to the notion of NL Queries. However, in this paper we focus on the second phase of our study in which we asked participants to imagine their own projects and to compose a set of useful NL queries. Our goal was to identify the scope of their interest as well as to analyze the terminology and structure of their queries. As a result of this activity we collected a total of 103 open queries.

3 Analysis

We classified the resulting queries according to their associated Software Engineering activities and also according to their semantic and syntactical characteristics related to query types and required functions.

3.1 Classification into Software Engineering Activities

Two of the researchers individually reviewed and categorized 25 of the queries. They then discussed the results and agreed upon an initial set of categories. One of the researchers then used this list to categorize the remaining queries. A few new categories were added. The results were then reviewed and small categories were merged or integrated with other categories. In the case of *project management*, *requirements*, and *test analysis* the process was repeated in order to create subcategories. Several queries cut across two or more categories. Sample categories with their query counts are shown in Table 1. Complete category list can be found here : *https://tinyurl.com/kj296j4*

3.2 Query Characteristics

The majority of queries represented a request to *list* something (49) e.g., "Which use cases have the higest risks?", to *count* something (28) e.g., "How many system requirements are there?", to return a *binary* answer (25) e.g., "Are any of the open fault logs related to an exception case?", or to provide a *rationale* (26) e.g., "Why is this requirement not completed?". Of these we determined rationale queries to be out of scope unless a specific field existed in which the rationales were documented.

Queries included specific functions. 13 queries involved **comparisons** such as identifying artifacts which exhibited either high or low values for some attribute. For example *Which use cases have the most test cases associated with them?*. 28 queries asked for results to be **aggregated**. For example, "How many code classes are needed for implementing a system requirement on average? (sic)". 13 queries included **negations**. For example *How many requirements are not yet implemented?*. There were 11 cases of **compound queries** such as "Can you please tell me on what dates tests were run, who the tester was on that day, how many test cases were run, and how many of them failed?" These queries often required initial results that needed to be fed as inputs into subsequent parts of the query.

Almost every query included **software engineering terms**. For example the query *How many new requirements have been* **elicited** *last month?* would require TiQi to understand that the *count of new requirements in the past month* should serve as a proxy for the answer. In other cases they used basic synonyms such as the term *backed up* as a synonym for *associated with*. Users also incorporated **domain terminology**. For example *Which test cases relate to the PCA pump?* To answer such queries requires knowledge of domain-specific synonyms, part-of, and is-a relationships. There were 3 such queries.

There were two types of queries that we classify as problematic: **Ambiguous** and **Out of Scope** queries. There were 12 queries that could be interpreted in multiple ways even by humans. For example *What has been done with my requirements?*. 34 of the queries were deemed to be out of scope. For example *Have we considered the design issues before making the decision to migrate to cloud?*. This query could only be answered if specific data had

Table 1: Queries by Category with Examples

Query Topic	**Examples (sic)**	**Count**
Design Analysis	Which designs are impacted by the environment assumption EA1	7
Fault Analysis	Is any of the open fault logs related to an exception case?	8
Personnel	Which programmers are more error prone in their code acording to the test results?	6

been captured about the design alternatives related to cloud migration. The issue of scope is therefore closely related to the availability of specific data.

4 Lessons Learned

The lessons learned from this empirical study will be useful for future enhancements and evaluation of TiQi. Most of the constructs (i.e. the problems of ambiguity, complex questions, aggregations, negations) had already been envisioned and are at least partially integrated into TiQi. These results served as confirmation for our previous conjectures.

A second primary contribution related to the scope of requirements engineering activities that TiQi must cover and the varied nature of the questions asked. As a result, we plan to augment TiQi's domain model to support specific types of activities - such as the ones highlighted here related to requirements, project management, and testing, as well as other envisioned areas of security, change management, and safety analysis. We plan to integrate the collected queries into an extended dataset for benchmarking purposes. Furthermore, each human-provided query will be augmented with variants representing negations, aggregations, and compound queries. It will also be reformulated into binary, list, and count query types. In this way, the results of this study will produce a greatly extended set of test queries for future experimentation.

5 Related Work

Other researchers have investigated the kinds of questions practitioners are interested in asking. Fritz et al.[4] interviewed 11 professional developers and identified 78 questions of interest primarily covering *source code*, *change sets*, *teams* and *work items*. They categorized the questions into groups such as *Who is working on what?*, *Changes to the code*, *Work item progress*, *broken builds*, *test cases*, and *references on the web*. However, their focus was on showing how each query could be serviced through the composition of information fragments and not on how the NL query could be dynamically translated into executable form. Other researchers have investigated specific types of queries such as those related to software development [5] or code change tasks [6]. In similar work, Bouillon et al. [7] conducted a survey of practitioners and identified 29 traceability usage scenarios such as *refining and detailing requirements* and *test coverage analysis of specification and code*. They grouped these scenarios into 6 different groups, each pertaining to a phase of the software development process.

All of these studies provide invaluable insights into the type of queries practitioners would

like to ask. They reflect quite different artifact types and interests from the more requirements focused ones identified by our study, which suggests the importance of analyzing queries from practitioners responsible for performing different types of tasks. We also observed that the queries reported in prior papers are more consistently formatted than ours. This is because the intent was to create a catalog of query types, rather than a collection of queries as-expressed by humans.

6 Conclusion

The primary contribution of this study is the collection of requirements-focused queries as expressed by our participants. An analysis of these queries identified types of questions and query characteristics which will be used in developing a more varied benchmark of queries. We close with an attempt to answer our earlier question of *what constitutes a fair dataset at any particular phase of the research project?* In initial phases of a project it seems appropriate to work with any available datasets. As the work will be at least partially judged by the quality and scope of such datasets they should be made available for inspection. However, as a project matures it is essential to move towards standardized benchmarking efforts so that ongoing work can be evaluated against realistic problem domains.

References

[1] P. Pruski, S. Lohar, R. Aquanette, G. Ott, S. Amornborvornwong, A. Rasin, J. Cleland-Huang, Tiqi: Towards natural language trace queries, in: IEEE 22nd Intn'l Requirements Engineering Conf., RE 2014, Karlskrona, Sweden, August 25-29, 2014, 2014, pp. 123–132.

[2] P. Pruski, S. Lohar, G. Ott, W. Goss, A. Rasin, J. Cleland-Huang, Tiqi: Answering unstructured natural language trace queries, Requir. Eng. (2015) To appear.

[3] S. Lohar, J. Cleland-Huang, A. Rasin, P. Mäder, Live study proposal: Collecting natural language trace queries, in: Joint Proceedings of REFSQ-2015 Workshops, Research Method Track, and Poster Track co-located with the 21st International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2015), Essen, Germany, March 23, 2015., 2015, pp. 207–210.

[4] T. Fritz, G. C. Murphy, Using information fragments to answer the questions developers ask, in: ICSE 2010, May 2-8, 2010, Cape Town, South Africa, ACM, 2010, pp. 175–184.

[5] A. J. Ko, R. DeLine, G. Venolia, Information needs in collocated software development teams, in: 29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 20-26, 2007, 2007, pp. 344–353.

[6] J. Sillito, G. C. Murphy, K. D. Volder, Asking and answering questions during a programming change task, IEEE Trans. Software Eng. 34 (4) (2008) 434–451.

[7] E. Bouillon, P. Mäder, I. Philippow, A survey on usage scenarios for requirements traceability in practice, in: Requirements Engineering:Foundation for Software Quality-19th International Working Conference, REFSQ 2013, Essen, Germany, April 8-11 2013.Proceedings, Springer, 2013, pp. 158–173.