Analyzing Natural-Language Artifacts of the Software Process

Maryam Hasan, Eleni Stroulia, Denilson Barbosa, Manar Alalfi
Dept. of Computing Science
University of Alberta
Edmonton, AB, Canada
{mhasan1, stroulia, denilson, alalfi}@ualberta.ca

***Abstract*—Software teams, as they communicate throughout the life-cycle of their projects, generate a substantial stream of textual data. Through emails and chats, developers discuss the requirements of their software system, they negotiate the distribution of tasks among them, and they make decisions about the system design, and the internal structure and functionalities of its code modules. The software research community has long recognized the importance and potential usefulness of such textual information. In this paper, we discuss our recent work on systematically analyzing several textual streams collected through our WikiDev2.0 tool. We use two different text-analysis methods to examine five different sources of textual data. We report on our experience using our method on analyzing the communications of a nine-member team over four months.**

I. Motivation and Background

Software developers, as they communicate with each other throughout the life-cycle of their projects, generate a substantial stream of textual data. The content of the developers' informal communications is a rich source of additional documentation that complements, and often elaborates on, the "formal" documentation that is produced to describe the software artifacts on which the team works.

Through email and chat, developers discuss the requirements of their software system, and make decisions about the design, the internal structure and the functionalities of the various modules in this system. A careful analysis of such text reveals valuable information about various aspects of the software life-cycle: the *background and expertise of developers* (e.g., *"I have used JUnit before"*), *traceability* links (e.g., *"To implement this requirement, we have to ..."*), the *roles and responsibilities* assumed by the various team members (e.g., *"You should be the tester for this package"*), and their *specific contributions* to the various project artifacts (e.g., *"I added method M to class C in order to make it respond to the user's choices of color"*). Furthermore, the informal nature of such communication channels can also offer ways of extracting more subtle information about the dynamics among a team's members (e.g., *"who assigns tasks", "who communicates the most/least and with whom", "who regularly disagrees with whom", "who asks the most questions"*).

The software research community has long recognized the importance and potential usefulness of this information and has developed several approaches to mining it, the most interesting of which are reviewed in Section IV. In this paper, we discuss our own recent work on systematically analyzing several sources of textual information collected through our WikiDev2.0 tool [1] as it is being used by an undergraduate software team over the course of a term-long project.

II. Method

The fundamental methodological assumption of our work is that, through their collaboration, teams develop a "shared understanding" about what they discuss in each of their communication channels and how. In order to exploit such information, we use two complimentary text-analysis methods, and apply them to five different sources of textual data. The first method is based on approximate and efficient analysis, at the lexical level, using the off-the-shelf lexical-analysis toolkit TAPoR [2], [3]. The second is much more accurate, albeit more expensive from a computational point of view, and focuses on the text at the syntactic/semantic level, relying on the Stanford parser [4]. In our work, we adopt these two text-analysis tools, apply them to five data sources and examine their results to gain insights about the software-development process and its products. The five textual-data sources are (a) wiki pages, (b) SVN comments, (c) tickets, (d) email messages, and (e) IRC chats.

A. TAPoR

TAPoR, the Text Analysis Portal for Research, is a web-based application developed by digital humanists to support a suite of lexical-analysis tools [2], [3]. The tools offered include among others (a) word counts and lists, (b) word co-occurrence (i.e., keyword-in-context), (c) word-clouds visualizations, (d) words' collocations within sentences and paragraphs, and (e) pattern extraction. TAPoR has several deployments around the world and is used to analyze the lexical properties of texts and collections. In our work, we have used TAPoR for two purposes. First, we used its "most-frequent-words" functionality to identify "important" keywords, which we then chose to examine in more detail, i.e., with the more precise syntactic/semantic analysis method, in the context of their sentences. Second, we applied the word-count and keyword-in-context functionalities to a systematically produced set of slices of our data set to gain insights about interesting trends in the information contained in the different data sources, as

26th IEEE International Conference on Software Maintenance in Timișoara, Romania

978-1-4244-8628-1/10/$26.00 ©2010 IEEE

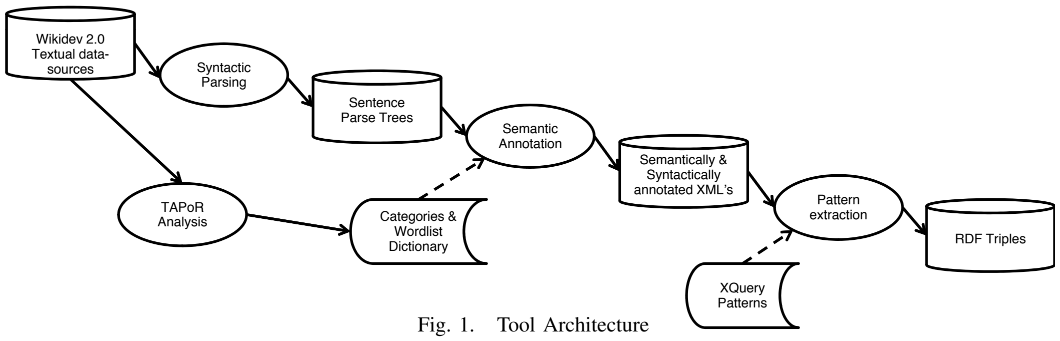

Fig. 1. Tool Architecture

```
Syntactic tags:

  I/PRP  used/VBD  java/NN  and/CC  Eclipse/NN  before/RB ./.

Dependency Relations:

nsubj (used-2, I-1)
dobj (used-2, java-3)
conj_and (java-3, Eclipse-5)
advmod (used-2, before-6)
```

Fig. 2. Parsing result for the sentence "I used Java and Eclipse before."

```
<S  Type="ticket-description"  ticketId="1"  sentId="127 Author="User1">
 I used Java and Eclipse before.
  <Verb  stem="use" ID="1" POS="VBN"  Relation="root"> Used
    <PRP ID="1"  POS="PRP"  Relation="nsubj"
         semanticTag="Developer" Name="User1">  I </PRP>
    <Noun ID="2"  POS="NNP"  Relation="dobj"
          semanticTag="language"> Java
           <Noun  ID="4"  POS="NNP"  Relation="conj_and"
          semanticTag="tool"> Eclipse </Noun>
    </Noun>
        <Adverb ID="5" POS="RB" Relation="advmod"> before </Adverb>
  </Verb>
</S>
```

Fig. 3. The sentence "I used Java and Eclipse before." after parsing and semantic annotation.

provided by the individual team members over the various stages of the project.

B. Syntactic/Semantic Analysis

TAPoR exemplifies a very lightweight analysis of natural-language text, focusing only at the "superficial" lexical level of the text. On the other end of the computational-complexity and "depth" spectrum, we have developed our own method for syntactic and semantic analysis, by integrating state-of-the-art computational-linguistic techniques with domain-specific knowledge, in order to extract useful pieces of information as RDF-style (Resource Description Framework [5]) triples. The extracted triples constitute an instance of a rich conceptual model of the domain that captures interesting relations between developers and software products. The triples are stored in a database in order for users to efficiently query them. Our method consists of (a) a syntactic parsing stage, (b) an annotation stage, where the syntax trees of the parsed sentences are annotated with semantic information, and (c) a pattern-matching query stage that extracts subject-predicate-object triples from the annotated parse trees, as depicted in Figure 1. More details of the process are described in the following subsections.

1) Syntactic Tagging and Parsing: For the first task of the process, namely syntactic tagging and dependency parsing, we use the Stanford Parser [4]. Before the parser can be applied, our data sources have first to be cleaned from any "noise" elements, such as pieces of source code inside the text, (wiki, html, and javadoc) markup tags, greetings, and signature and quoted parts from earlier mail messages. Next, we fix possible punctuation errors in the name of domain-specific terms. Finally, we split the textual data into the separate sentences.

The separated sentences are parsed by the Stanford parser, whereby each word is assigned a syntactic tag such as noun, verb, adjective and adverb. Furthermore, the parser also provides the grammatical relationships between word pairs such as subject, object, possession, etc. The parser presents the dependency relations as triples: name of the relation, governor and dependent elements. Figure 2 presents the syntactic tags and the dependency relations created by the Stanford parser for the sentence "I have used Java and Eclipse before".

2) Term Extraction and Semantic Annotation: Syntactic parsing establishes grammatical relations among the words of a sentence. In the next phase of our method, the task becomes to annotate the sentence words with domain-specific semantics. This semantic-annotation phase is implemented as a process of "attaching" terms from a domain-specific vocabulary, to, as many as possible, corpus words. Through this association of sentence words with domain-specific terms, the process attempts to clarify the semantics of these words. The question then becomes to identify the vocabulary terms. In our work we have used the TAPoR word-frequency service to identify these terms from our corpus such as project tasks, project artifacts, and the names of the tools. The basic underlying idea is that interesting and meaningful (for the domain) terms are likely to occur with higher frequency in a corpus of this domain. In addition to TAPoR's most frequent words, we also consider a set of terms that are already part of the WikiDev2.0 database, such as the (anonymized) names of the teams' members, the names of the developed files, and programming languages the teams use, and the IDs of the teams' tickets and code revisions. After aggregating these two sets of terms, we created a hierarchically organized vocabulary for the software-development domain. The vocabulary includes the following categories and their corresponding words.

- *Users* = { *User1, User2, ... , User9* }.
- *Programming Languages* = { *Java, PHP, XML, ...* }.

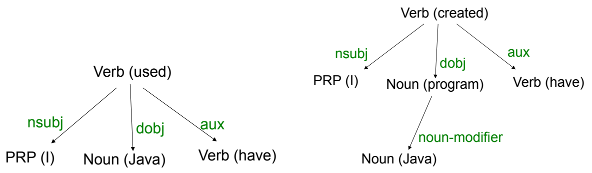

Fig. 4. Dependency paths in the XML tree for the sentences "I have used Java." and "I have created a Java program."

- *Tools* = { *Eclipse, Bugzilla, IBMJazz, ...* }.
- *Tickets* = { *ticket1, ticket2 , ...* }.
- *Revisions* = { *revision1, revision2, ...* }.
- *Action Verbs* = { *create, debug, implement, fix, make, ...* }.
- *Project Tasks* = { *visualization, documentation, user interface, testing, ...* }.
- *Project Artifacts* = { *class, method, database, script, ...* }.

Given this domain vocabulary, the semantic-annotation process involves the annotation of nouns and noun phrases, as identified by the syntactic-parsing phase before, with a *category:term* tag, such as *Users:User1* for example.

At this stage of the process, each sentence has been transformed into a syntactically and semantically annotated tree, represented in XML. This representation is much more amenable to querying than the original sentence. As shown in Figure 3, each such annotated XML tree includes the following pieces of information:

- Dependency relations between words, such as "dobj" between "Java" and "used";
- Syntactic tags of the words, such as "NNP" for the word "Java";
- Semantic annotation of the domain-specific terms, such as "language" for the term "Java";
- Stem of the verbs, such as "use" for the verb "used";
- Type of the sentence, such as "ticket-description"; and
- Author of the sentence, such as "User1".

Figure 3 shows the XML document, resulting from the parsing and semantic annotation of the sentence "I used Java and Eclipse before".

3) Pattern Extraction: The next phase of our method involves the matching of "patterns" against the semantically and syntactically annotated XML trees of our sentences, in order to extract knowledge in the form of RDF triples.

Each RDF triple expresses a semantic relation between two entities. Our pattern-extraction process attempts to recognize instances of these semantic relations as "syntactic paths" that connect a verb with two annotated entities close to it. The relation verb along with the associated entities becomes "a relation triple", which can indicate potentially a relevant semantic relation. Essentially, the syntactic dependencies recognized by the Stanford parser provide the basis for semantic relations between the semantically annotated entities.

The proposed approach for the triple extraction traverses the annotated XML trees and analyzes the syntactic dependencies and the semantic annotations in order to find predicate-argument patterns among the domain concepts. The pattern-extraction task is implemented as a rule-based process. We defined the following two rules to search for the predicate-argument patterns that exist in the annotated XML trees and extract triples from the annotated XML trees.

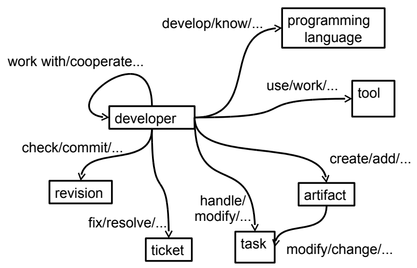

Fig. 5. The semantic relations of the RDF triples produced by the pattern-extraction process.

Rule 1: If a verb V is associated with two different entities E_i and E_j which satisfy the condition $[Subj(V, E_i) \wedge Rel(V, E_j)]$, then V is identified as a *relational verb* between entities E_i and E_j. Based on the dependency relations identified by the parser, $Rel(V, E_j)$ can be either an "object relation" or a "prepositional relation". For example, in "I have used JUnit" there is an object relation $dobj(use, JUnit)$ while in "I have worked with JUnit" there is a prepositional relation $prep\_with(use, JUnit)$.

Rule 2: If a verb V is associated with three entities E_i, E_j and E_k, satisfying $[Subj(V, E_i) \wedge Rel_1(V, E_j) \wedge Rel_2(E_j, E_k)]$, then V is identified as a *relational verb* between the two entities E_i and E_k. Based on the dependency relations identified by the parser, the relation $Rel(V, E_j)$ can be an "object relation" or a "prepositional relation" and the relation $Rel_2(E_j, E_k)$ can be a "prepositional relation", a "conjunction relation" or a "noun-modifier relation".

Intuitively, Rule 1 covers explicit relations directly expressed in the text, while Rule 2 captures implicit relations, such as indirect objects and long distance dependence relations (see Figure 4). We apply both rules to find relevant semantic relationships between *types* of entities, instead of specific entities themselves. Therefore, we are looking for the triples of the type $\langle category, semantic\_relation, category \rangle$. The list of our target triples is shown in the model of Figure 5.

Based on the above two rules, we defined a set of patterns, implemented in XQuery, which we apply to the annotated XML trees to retrieve interesting RDF triples. XQuery is a powerful query language designed to query XML data. The answers returned by running XQuery on annotated XMLs are the instances of our target triples. At this point, we have generated fourteen patterns for extracting the target triples using the defined rules.

III. Empirical Results

In the rest of this paper we demonstrate our approach by studying the behavior of a student team who used WikiDev2.0 for their project. The project duration was four months, during which a distributed team with nine members used a customized version of the WikiDev2.0 collaboration tool as a platform to interact with each other, and to assist them in their assigned project development tasks. In this experiment, and based on the

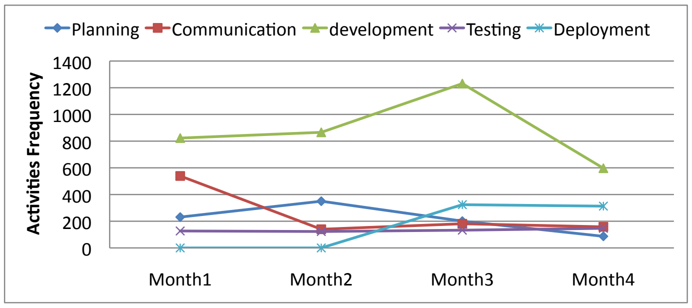

Fig. 6. Trends of the team activities throughout the project duration

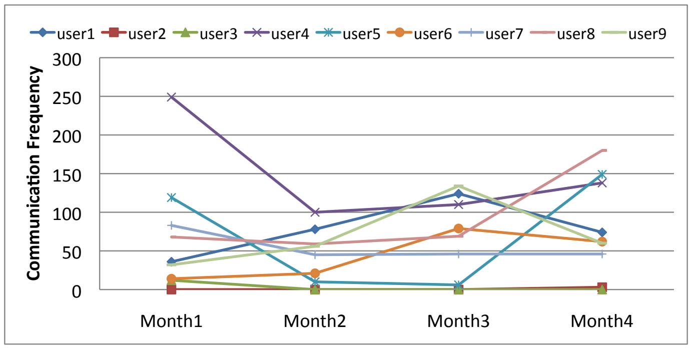

Fig. 7. Trends of team members' communication throughout the project duration

collaborative information collected by the WikiDev2.0 tool, we used two methods to systematically analyze the textual data sources gathered during the project lifetime. Results from those analyzes are presented in Sections III-A and III-B.

A. Lexical Analysis using the TAPoR service

We have systematically generated, from the Wikidev 2.0 database, multiple textual datasets originating from the different types of tools integrated in the system: (a) email messages, (b) IRC chats, (c) SV commit comments, (d) ticket descriptions, and (e) wiki pages. We "sliced" these datasets based on each of their authors (i.e., individual team members) and based on time. We then applied the TAPoR service on these datasets, and categorized the most frequent words based on their relevance to the project life-cycle (communication, planning, development/implementation, testing and deployment). For instance, words like *"meeting, team, progress, messages, todo"* were classified in the communication category, while words relevant to the project artifact such as *"XML, UML, Parser,.."* were classified in the development/implementation category, and *"Screencasts, documentation, attachments, upload"* were classified in the deployment category. The frequencies of these words are reported in Figures 6 and 7.

The graph shown in Figure 6 presents the trend of team members' activities during the project life-cycle. Some interesting information can be gleaned from this graph. The "development/implementation" process is the dominant activity during the project life-cycle. The level of communication between team members reached its peak in the first month and then decreased to a steady level during the following months, until the end of the project. The "testing" process spanned all the project lifetime with the same level of activity.

	SVN comment	Ticket	Email message	IRC chats	Extracted Samples
Number of Sentences	353	169	484	3130	
Number of annotated XMLs	346	161	461	3018	
Number of Triplets	154	77	165	830	
<developer, use/..., language>	10	6	15	85	<user6 focus on Java>
<developer, work/..., tool>	16	4	20	74	<user5 used Eclipse>
<developer, create/..., artifact>	82	34	55	210	<user5 handle XMIParser>
<developer, handle/..., task>	12	5	10	68	<user8 work on UI>
<developer, fix/..., ticket>	4	5	4	16	<user9 fixed bug5>
<developer, check/..., revision>	5	8	0	3	<user6 done revision53>
<artifact, change/..., task>	25	12	7	64	<UMLHandler modify Xml>
<developer, cooperate/..., developer>	0	3	56	310	<user6 & user17 focus on parser>

Fig. 8. Syntactic/Semantic Analysis Results from four data sources.

Figure 7 shows the trend of communication for each member of the team. Interestingly, "user4" was dominating the communication during the first month, and his/her communication level decreased a bit for the second month but his/her level of communication remains the same for the following months and almost stay higher than all other user communications. After observing these trends, we identified this user as the team "manager". Other valuable information can be conveyed from graphs generated using the same approach for different datasets, such as team members roles, the relation between team member activity and the quality of his/her contribution to the project and other team members.

B. Syntactic/Semantic Analysis Results

We applied the syntactic/semantic analysis process on the first four datasets above. Figure 8 summarily reports on the results of this activity. The first three rows show that there is a significant semantic relations between domain-specific terms. From these semantic relations we can find several useful pieces of information, as discussed below.

1) Expertise of developers: Triples about language or tool showed that User6 and User9 mostly used PHP, MySQL and PostgreSQL, while User5 and User7 used Flex, Eclipse, and UML2. Some sample sentences are *User9 tried to run the Php code*, and *User5 has started learning Php*.

2) Responsibility of developers: We looked for triples mentioning project tasks. We found that User5 mostly worked on testing, User8 worked on documentation and testing, User7 worked on user interface, User9 worked on designing, and User5, User6, and User7 worked together on screencast. Some sample sentences are *User7 neaten up the UI along the side of UML display*, and *User8 take care of documentation*.

3) Developers' contributions to the project: By searching for the triples about developed classes, we found out that User5 and User6 worked with XMIParser, User6 and User9 worked on UMLHandler, User6 worked on UMLViewer, and User8 worked with Wikiroamer and WikiviewFactor. Here are some sample sentences of this group: *User5 handled associations in XMIParser*, *User8 changes wikiroamer and wikiviewfactor*.

4) Developers' relationships: The triples mentioning two developers show that User4 had the most relationships and worked with all developers, and User1 and User3 got the second and third place. User5 mostly worked with User6, and User9 worked with User6 and User8. Sample sentences include *User6 modify User5 's Php file to allow ...*, and *User6 and User9 should focus on preparing parser.*.

IV. Related Work

There has been a significant amount of work (a) on natural-language analysis for understanding the life-cycle of software projects, and (b) on understanding the communication artifacts around software projects. Würsch and his colleagues [6] developed an approach to answer program-comprehension questions, such as "what are the subclasses of a class?", "what fields are declared as this type?". Their work is similar to ours, in that they model software data with an OWL ontology and they represent the source code information as an RDF graph based on this ontology. However, WikiDev2.0 integrates more data sources, especially ones reflecting the developers' communication channels such as email, IRC and wiki pages, which their tool ignores.

Yu and his colleagues [7] and Bird et al. [8] adopt a social-network model for representing the interactions of open-source software developers. However, they do not actually examine the text of the communication but only the connectivity of the senders and recipients.

Rigby and Hassan [9] used LIWC to examine the mailing list of Apache httpd server developers. LIWC is similar to TAPoR, not able to do substantial text analysis.

Yoshikawa and his colleagues [10] proposed a technique to find traceability links between a natural-language sentence describing features of a software product and the source code of the product using a domain ontology.

More recently, Fritz and Murphy [11] identified 78 interesting questions about software development and developed a tool to answer these questions (actually 8 of them) by composing information fragments, originating primarily from traditional repository-mining methods. The tool also examines textual data from Wikis but it does not do any substantial syntactic/sementic analysis; rather it relies on recognizing IDs and names. A similar approach is used by Codebook [12] which uses crawlers to extract information from a project repository. With respect to text, it simply recognizes names withing textual objects and infers that the object in question "mentions" the named individual.

V. Conclusions and Future Work

In this paper, we discussed the initial results of our work on lexical, syntactic, and semantic analysis of textual data produced during the life-cycle of a software project. The informal communications among the team developers and their documentation of their artifacts and activities in email messages, IRC chats, SVN commit messages, ticket descriptions and wiki pages contain valuable information about the issues the team members faced during their work and the decisions they made. Thus, extracting this information can provide valuable insight into the collaboration style of the team, the relative contributions of each member and the relation of the various software artifacts to the original project requirements. Our empirical results to date, indeed, validate this hypothesis. We are currently working towards (a) extending the suite of RDF-triple patterns, and (b) developing a domain-specific query language (based on our underlying conceptual model of Figure 5) for flexible question answering on the project lifecycle. What is of particular interest in the future is to see how the information from more traditional repository-mining approaches can be combined and complemented with information extracted from textual data, and whether the text can provide further evidence to support (or possibly refute) the information inferred by these other approaches.

Acknowledgment: The authors would like to acknowledge the generous support of iCORE, AIF, NSERC, and IBM.

References

[1] K. Bauer, M. Fokaefs, B. Tansey, and E. Stroulia, "WikiDev 2.0: discovering clusters of related team artifacts," in *CASCON '09: Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*. New York, NY, USA: ACM, 2009, pp. 174–187.

[2] H. Vashishtha, M. Smit, and E. Stroulia, "Moving text analysis tools to the cloud," in *The IEEE 6th World Congress on Services (SERVICES 2010), USA*, 2010 (to appear).

[3] "TAPoR: Text Analysis Portal for Research," last access June 26, 2010. [Online]. Available: http://hypatia.cs.ualberta.ca/tapor/

[4] "The Stanford Parser: A statistical parser," last access June 26, 2010. [Online]. Available: http://nlp.stanford.edu/software/lex-parser.shtml

[5] "Resource description framework (rdf)."

[6] M. Würsch, G. Ghezzi, G. Reif, and H. Gall, "Supporting developers with natural language queries," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, 2010, pp. 165–174.

[7] L. Yu, S. Ramaswamy, and C. Zhang, "Mining Email Archives and Simulating the Dynamics of Open-Source Project Developer Networks."

[8] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*. New York, NY, USA: ACM, 2006, pp. 137–143.

[9] P. C. Rigby and A. E. Hassan, "What Can OSS Mailing Lists Tell Us? A Preliminary Psychometric Text Analysis of the Apache Developer Mailing List," in *Fourth International Workshop on Mining Software Repositories, MSR 2007 (ICSE Workshop), Minneapolis, MN, USA, May 19-20, 2007, Proceedings*, 2007, p. 23.

[10] T. Yoshikawa, S. Hayashi, and M. Saeki, "Recovering traceability links between a simple natural language sentence and source code using domain ontologies," in *25th IEEE International Conference on Software Maintenance (ICSM 2009), September 20-26, 2009, Edmonton, Alberta, Canada*, 2009, pp. 551–554.

[11] T. Fritz and G. C. Murphy, "Using information fragments to answer the questions developers ask," in *ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*. New York, NY, USA: ACM, 2010, pp. 175–184.

[12] A. Begel, K. Y. Phang, and T. Zimmermann, "Codebook: discovering and exploiting relationships in software repositories," in *ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*. New York, NY, USA: ACM, 2010, pp. 125–134.