Using Information Fragments to Answer the Questions Developers Ask

Thomas Fritz and Gail C. Murphy
Department of Computer Science
University of British Columbia
Vancouver, BC, Canada
{fritz,murphy}@cs.ubc.ca

ABSTRACT

Each day, a software developer needs to answer a variety of questions that require the integration of different kinds of project information. Currently, answering these questions, such as "What have my co-workers been doing?", is tedious, and sometimes impossible, because the only support available requires the developer to manually link and traverse the information step-by-step. Through interviews with eleven professional developers, we identified 78 questions developers want to ask, but for which support is lacking. We introduce an information fragment model (and prototype tool) that automates the composition of different kinds of information and that allows developers to easily choose how to display the composed information. In a study, 18 professional developers used the prototype tool to answer eight of the 78 questions. All developers were able to easily use the prototype to successfully answer 94% of questions in a mean time of 2.3 minutes per question.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Human Factors, Design

Keywords

Information fragments, Human-centric software engineering, Programming tools

1. INTRODUCTION

A typical day of work requires a software developer to answer a variety of questions (e.g., [9, 14]). Some of these questions are easy to answer because they focus on one kind of information and have little ambiguity. An example of one of these questions is "Where is this method called [...]?" [14], which focuses on source code and typically has a well-defined meaning. Other questions are harder to answer because they require integrating multiple kinds of information and can be interpreted in many different ways. An example of one of these question is "What have my coworkers been doing?"[9]. Depending upon the developer asking the question, an answer may involve only information from a revision system to determine what code is changing or it may also involve information from a bug repository to help explain why the change is occurring.

From our own experience talking with developers, we believed that a number of the questions of interest are of the second form. To investigate the range of questions that span multiple kinds of information, we interviewed eleven professional software developers (Section 3), finding 78 questions of interest to them that have the characteristic of requiring multiple kinds of information to answer. We also found that even though many of these 78 questions sound similar, there can be substantial variation in how a developer interprets a question.

Although developers want to ask these questions, existing approaches provide little support for answering them (Section 2). Consider the question "What have my coworkers been doing?". To answer this question, existing development environments require a developer to explicitly follow links that exist between pairs of the kinds of information of interest; each kind of information is displayed in its own view. Specifically, a developer has to select select a bug that describes ongoing work in one view, follow the link to a set of changes that describe what work was done shown in another view and then finally bring up the changed source code in a third view to know specifically what work was conducted. These steps must be iteratively performed for *each* change set associated with the bug. By the time the developer follows all desired links, it is quite possible that he has forgotten the question of interest! More importantly, only one chain of linked information can be followed at a time; if multiple chains are needed to answer the question, such as the one we are considering that is oriented at multiple members of a team, the burden is placed on the developer to mentally and manually correlate the information reached.

An alternate solution is to provide a query language that allows a developer to specify explicitly how the information should be integrated (e.g., [12, 8]). The problem with this solution is that the developer must know or learn the query language and must express explicitly how the information should be integrated. These approaches mean that the de-

ICSE '10, May 2-8 2010, Cape Town, South Africa
.

veloper needs to know about and express in the query how the bug information maps to the change set information and how the change set information maps to the source code.

We introduce a model that automates the integration of different kinds of project information by using the structure of the information (Section 4). Our model has two advantages. First, the integration of information, which we call composition, is automated; as a result, a developer need only indicate which information to integrate rather than how to integrate it. Second, our model separates the composition from the presentation of information. This separation allows the developer to express a large variety of questions and to tailor the composed information to his personal interpretation of the question at hand. Using our model, we have been able to express all 78 questions determined through the interview study.

Expressibility of a large set of questions is necessary to make the model useful. To ensure the model can be applied by developers, we implemented a prototype tool (Section 6) and conducted a case study (Section 7) involving 18 professional developers. We found that without needing to know the details of how the composition of the information occurs in the model, the developers were able to easily use the prototype to successfully answer 94% of eight questions posed in a mean time of 2.3 minutes per question.

This paper makes three contributions.

- It presents 78 questions that developers ask; these questions reveal the range of interpretations developers desire for compositions of project information.
- It introduces the information fragment model that simplifies the answering of these 78 questions compared to previous approaches.
- It demonstrates that the model can be easily used by developers through a study with 18 professional developers.

2. RELATED WORK

A variety of approaches have been developed to help users navigate through complex information spaces by following information links. Feldspar allows a user to follow association links between different kinds of information, such as emails and files, to find information of interest [2]. Haystack lets a user specify views for different kinds of information, but still requires the user to step manually across the views to correlate information [13]. JQuery provides support specific for software development, allowing a developer to step through structural links between source code elements within a single view, one at a time [7].

Other approaches to helping retrieve information support the user in expressing, rather than following, information links. These approaches involve a query language that requires the user to explicitly form links between the information. Some, such as the relational views by Linton [11] and the approach by Paul and Prakash [12], are based on a relational algebra. Others, such as CodeQuest [5] provide a logic-based query language. Kiefer and colleagues [8] broaden the range of information that can be retrieved, covering source, revision and bug information, with the use of the iSPARQL query language that is based on a fixed ontology. All of these approaches require the user to express as part of the query how the information links together.

Alternatively, some systems have automatically linked different kinds of software development project information. Hipikat uses a fixed schema to mine information from software project repositories and provides a developer multiple entry points into the created web of information [15]. The STeP_IN system extends Hipikat by including information on programmers [16]. Deep Intellisense automatically displays information relevant to a selected source code element such as change events and the people involved[6]. These approaches are oriented at accessing one view of integrated information. Hipikat, for instance, can recommend artifacts related to a provided starting artifact. In contrast, our approach focuses on allowing the user to compose and view different kinds of project information to support the many questions that arise during a work day.

Ferret by de Alwis and Murphy [3] supports the composition of different perspectives—called spheres in their approach—on similar information. For example, a sphere representing source code can be composed with a sphere representing dynamic information about the system to help a developer investigate such properties as which calls between methods are actually occurring in a run of the system. Their approach focuses on matching similar elements between spheres whereas our approach aims to support the composition of different kinds of information for which links can be automatically determined. Our model also supports the user in varying the presentation of composed information to enable a suitable interpretation for the task at hand.

The work presented in this paper augments and extends concepts we presented earlier [4] with empirical evidence for both the problem and the effectiveness of our approach and by presenting details of the underlying model.

3. DEVELOPERS' QUESTIONS

Earlier studies have considered questions developers ask during a software development project. Four recent studies provide the most comprehensive question catalogs. Two of these catalogs focus largely on questions about source code. Sillito and colleagues described 44 questions that developers ask when programming [14]. De Alwis and Murphy identified 36 questions from literature, blogs and their own experience that deal largely with source code [3].

The other two studies discuss questions about a broader set of software development activities. Ko and colleagues report on 21 types of questions determined by shadowing professional developers at work [9]. LaToza and colleagues propose 19 problems from their own experience as software developers and present the results of a survey of professional developers about the seriousness of each problem. A number of these question types and problems require multiple kinds of information to answer and have multiple possible interpretations. Neither of these two studies talks about the ambiguity that lies in the interpretation of questions such as "What have my coworkers been doing?". While abstraction helps in understanding the question space, it is the detailed questions that the developer needs answered.

To determine the specific questions developers need answered that span different kinds of information, we interviewed eleven experienced software developers. We learned about 78 questions that the developers face and about the differences in the interpretation of very similar questions. To ease presentation, we use the term *domain* in the remainder

of the paper to refer to information of a particular kind, such as the domain of bug reports or the domain of source code.

3.1 Subjects and Interview Process

We conducted open interviews with eleven professional developers from three different sites of one company. These developers represented a spectrum of roles and experience. The roles ranged from junior developer to team leads. The experience of these developers ranged from 1 to 22 years.

In a pilot for this study, we asked developers directly about questions requiring information from multiple domains that occur for them during development. Through the pilot, we found that the developers had difficulty understanding the kinds of questions that might meet this criteria.

Thus, we changed the interview method to start with a brief demonstration of a small tool we built that enabled the composition of source code, change sets, work items[1] and team information. Our intent in showing this prototype was to stimulate developers to think about questions that might arise when such composition is possible. We asked the developer to describe such questions and how he would want them answered. Throughout the interview, we adapted our questions to the scenarios the developer described.

Each interview session was between 15 and 60 minutes depending on the developer's availability and responsiveness. Throughout each session, the interviewer (the first author of this paper) took handwritten notes. We parsed our notes looking for the questions developers stated, as well as the meaning of the questions to the developers.

3.2 Interview Results

From the eleven interviews, we determined 78 questions that span across multiple domains. Only one of the 78 questions was stated by two different developers, the other 77 questions were stated by one developer each. For space reasons, Table 1 lists 46 of the 78 questions.[2] These questions span eight domains of information: source code (SC), change sets (CHS), teams (T), work items (WI), web sites and wiki pages (WW), comments on work items[3] (CO), exception stack traces (ST) and test cases (TC). Table 1 shows which domains, based on developer statements, are needed to answer each question. For ease of reading, we have grouped the questions into categories that roughly correspond to the domains needed to answer the questions. Some questions are annotated with a *, which denotes questions explicitly stated by at least one developer. All other questions are interpretations we have made as the interviewed developer either gave a long statement to describe the scenario or more context was needed to be able to state the question. The majority of the questions (51 of the 78) are based on the domains we presented in our demonstration at the start of the interview; the remaining questions incorporate a domain we did not present.

In our interviews, we focused on the variety and richness of questions rather than their frequency. From the interviews, we determined that some of these questions are looked at a couple of times a day, such as "What classes have been changed?"(21). On other questions, developers spend a lot more time. One developer stated that he spends 70% of his time on the question "Which conversations in work items have I been mentioned?"(46) to make sure he does not block other developers from working. Without an extensive field study it is not possible to measure exactly how often each of these questions arises throughout a work day or week, or how much time a developer spends on each of these questions.

Some of the questions sound very similar in their wording, but their answer differs depending on the individual interpretation of the developer. For example, even though the two questions "Who is working on what?"(1) and "What have people been working on?"(6) seem very similar, the answer to question (1) uses the four domains SC, CHS, T and WI, whereas the answer to question (6) uses only the two domains T and WI.

A lot of the questions also require the same domains to be answered. However, the developers expressed wanting the presentation of the answer in different ways. "Who is working on what?"(1) and "What classes has my team been working on?"(11), both require information on SC, CHS, T, and WI to lead to a meaningful answer. However, based on developers' statements, "Who is working on what?" should display the team members first and the work they have done below with the changed source code last, whereas "What classes has my team been working on?", the developer wanted to see the elements in the opposite order. In Table 1, (1)-(4) and (8)-(13) are two blocks of such questions, where the questions require the same domains but different presentations.

A last subtlety comes with the selection of the actual information. Similar questions often just differ in small details of the information of relevance. However, this difference influences the size of the result and the ease of interpreting it. For example, "What has changed between two builds and who has changed it?"(14) as well as "Who has made changes to my classes?"(15) require information from the same domains. The latter question is however only looking for classes the developer is working on, whereas the first (14) refers to all classes of the project.

3.3 Threats

Because of difficulties we had getting subjects to articulate questions involving multiple domains, in this study, we stimulated developers to think about such questions by demonstrating a very early version of the tool we subjected to later testing (Section 6). This demonstration may have biased developers to state questions answerable with our approach. We believe this threat to validity is small as other researchers have found similar questions (e.g., [9, 10]), adding credence to these being questions developers ask when working. The list of questions we present is partial and is not representative of all multi-domain questions a developer may ask.

4. INFORMATION FRAGMENT MODEL

To enable a developer to answer the wide range of questions that can arise, we introduce a model that supports the composition and presentation of information fragments.

4.1 Example of Use

We introduce our model by showing how it can be used to answer the question, "What have people been working on?"(Question 6 in Table 1). We use a simplified version of this example to present our approach succinctly.

[1]Work items are similar to bugs, issues or tasks.

[2]http://www.cs.ubc.ca/labs/spl/projects/infofrags.html

[3]Based on the developers' statements, we considered emails as being equivalent to comments.

Table 1: Developers' Questions and the Operators and Domains for Desired Answers (*: *question explicitly stated by a developer*, ***id*: *identifier matching*, ***t***: *text matching*)**

Question	***Operator***	Source Code	Change Sets	Teams	Work Items	Comments	Web/Wiki	Stack Traces	Test Cases
Who is working on what (people specific)									
1. Who is working on what?*	*id*	X	X	X	X				
2. What are they [coworkers] working on right now?*	*id*	X	X	X	X				
3. How much work [have] people done?*	*id*	X	X	X	X				
4. Who changed this [code], focused on person?*	*id*	X	X	X	X				
5. Who to assign a code review to? / Who has the knowledge to do the code review?	*id*	X	X	X	X				
6. What have people been working on?*	*id*			X	X				
7. Which code reviews have been assigned to which person?*	*id*			X	X				
Changes to the code (code specific)									
8. What is the evolution of the code?	*id*	X	X	X	X				
9. Why were they [these changes] introduced?*	*id*	X	X	X	X				
10. Who made a particular change and why?	*id*	X	X	X	X				
11. What classes has my team been working on?*	*id*	X	X	X	X				
12. What are the changes on newly resolved work items related to me?	*id*	X	X	X	X				
13. Who is working on the same classes as I am and for which work item?	*id*	X	X	X	X				
14. What has changed between two builds [and] who has changed it?*	*id*	X	X	X					
15. Who has made changes to my classes?	*id*	X	X	X					
16. Who is using that API [that I am about to change]?*	*id*	X	X	X					
17. Who created the API [that I am about to change]?*	*id*	X	X	X					
18. Who owns this piece of code? / Who modified it the latest?*	*id*	X	X	X					
19. Who owns this piece of code? / Who modified it most?*	*id*	X	X	X					
20. Who to talk to if you have to work with packages you haven't worked with?	*id*	X	X	X					
21. What classes have been changed?*	*id*	X	X						
22. [Which] API has changed (to see which methods are not supported any more)?*	*id*	X	X						
23. What's the most popular class? [Which class has been changed most?]*	*id*	X	X						
24. Which other code that I worked on uses this code pattern / utility function?	*id*	X	X						
25. Which code has recently changed that is related to me?	*id*	X	X						
26. How do recently delivered changes affect changes that I am working on?*	*id*	X	X						
27. What code is related to a change?	*id*	X	X						
Work item progress									
28. What is the recent activity on a plan item?	*id*				X	X			
29. Which features and functions have been changing?*	*id*				X	X			
30. Has progress been made on blockers (blocking work items) in your milestone?	*id*				X	X			
31. Is progress (changes) being made on plan items?	*id*		X		X				
Broken builds									
32. What caused this build to break? (Which change caused the stack trace?)	*id*	X	X					X	
33. Who caused this build to break? (Who owns the broken tests?)	*id*	X	X	X					X
34. Who changed the test case most recently that caused the build to fail?	*id*	X	X	X					X
35. Which changes caused the tests to fail and thus the build to break?	*id*	X	X					X	X
Test cases									
36. Who owns a test case? (Who resolved the last work item that fixed the test case?)	*id*	X	X	X	X				X
37. How do test cases relate to packages/classes?	*id*	X							X
References on the web									
38. Which API has changed (check on web site)?	*t*	X					X		
39. [Is an entry] in newsgroup forum addressed to me because of the class mentioned?*	*t*	X					X		
40. What is coming up next week [for my team]? [What is my team doing?]*	*t*			X			X		
41. What am I supposed to work on [plan on wiki]?*	*t*			X			X		
Other Questions									
42. How is the team organized?*	*id*	X	X	X					
43. Who has made changes to [a] defect?*	*id*		X	X	X				
44. Who has made comments in defect?*	*id*			X	X	X			
45. [What is] the collaboration tree around a feature?*	*id*			X	X	X			
46. Which conversations in work items have I been mentioned?*	*t*,*id*			X	X	X			

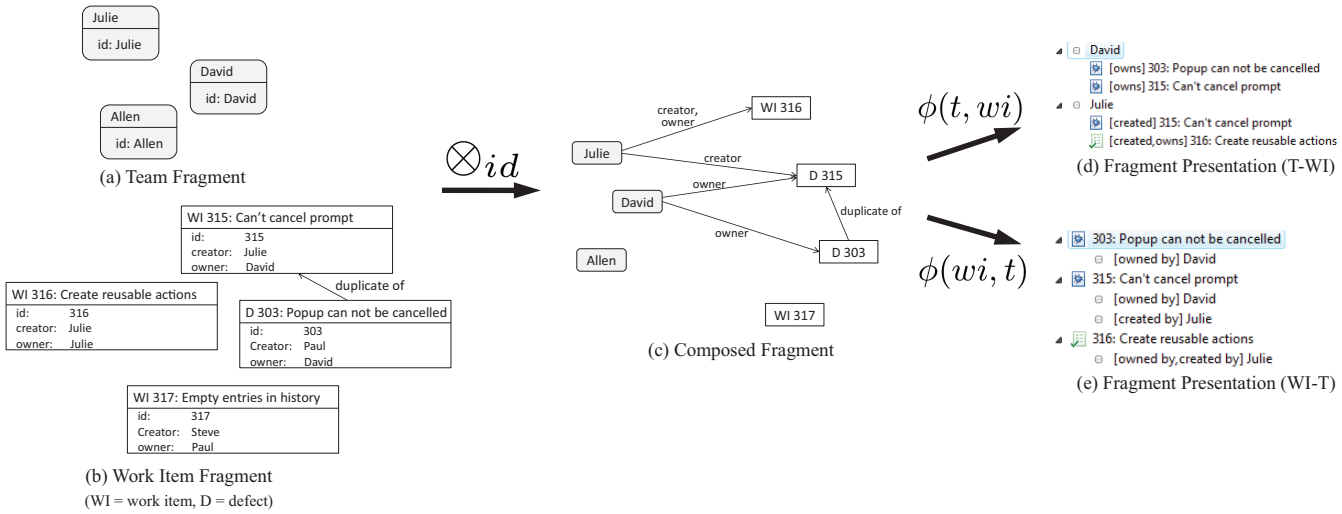


Figure 1: Approach to Answer the Question "What have people been working on?"

Consider a software developer Sue who gets back to work after a week of holidays. When she starts working, she wants to know what other developers have been working on while she was away. For this question, Sue is interested in two different information domains: teams and work items. From these two domains, Sue is interested in certain subsets, which we refer to as *information fragments*. One information fragment consists of the developers of her team (Figure 1(a)), and the second consists of the work items on which people have been working (Figure 1(b)). In our approach, information fragments are modeled as graphs with nodes and edges. Nodes represent uniquely identifiable items with properties; at least one property is specified as a unique identifier. For example, a work item includes an identifier, a creator and an owner property. Edges represent relationships between items, such as a "duplicate-of" relationship between two work items, stating that a work item is a duplicate of another work item.

Each fragment in isolation is not meaningful to Sue. However, by composing these two fragments, Sue can create the context that allows her to answer the question at hand. Our approach provides *composition operators* to compose information fragments. When Sue composes the two information fragments using $\otimes_{id}$ a new information fragment (Figure 1(c)) is created from the input fragments with new edges introduced between nodes based on the nodes' properties. For example, in the new information fragment, a new "owner" edge is created between David and the defect 303, because the owner property of the node representing defect 303 matches the identifier of the node representing David. Another edge is created between Julie and work item 316 because Julie is the owner and creator of 316.

As the answer to her question, Sue likes to see the work items ordered by developers. Therefore, she chooses a presentation that orders nodes of the team fragment (t) above the ones from the work item fragment (wi); this presentation is referred to as a projection ($\phi(t, wi)$). Figure 1(d) shows the result of this projection. In this presentation, her teammate Allen does not show up in the final presentations as he did not work on, and is thus not connected to, any work item.

Alternatively, Sue might be interested in seeing her team members in context of the work items on which they have been working. In this case, she would use a projection $\phi(wi, t)$. This projection shows the work items her fellow team members have been working on first and the developers below (Figure 1(e)). By separating the presentation from the composition of the information, either interpretation of the question is easily supported.

4.2 Information Fragments

An information fragment is a subset of development information for the system of interest. In our model, an information fragment is modeled as a graph $F = (V, E, l_F)$, with a set of labeled nodes V, a set of directed labeled edges E and a function $l_F : V \mapsto FS$ that assigns each node a set of information fragments FS.

Each node represents a uniquely identifiable item of information, such as defect 303 or team member Julie. Each node also has a *type*, such as defect, work item or team member, and, based on the type, a set of properties that describe the node. Finally, each node belongs to a *domain*, such as work items or teams, where a domain is a set of types with type sets of different domains being disjoint. At present, each type of a domain has the same properties. An exemplary list of domains, types and properties, is given in Table 2; for instance, a defect of the work item domain has an identifier and information on the creator and owner of the defect, among others.

Edges represent relationships between these uniquely identifiable items. An edge can be explicit, such as a method call, or implicit, such as a relationship between a work item and a team member that can be inferred from the nodes. An edge can also contain properties that describe the edge; for instance, an edge based on textual similarity between two work items can have the amount of similarity of the two nodes as a property.

A base fragment is a special case of an information fragment that contains nodes of one domain only. The fragments in Figure 1(a) and (b) are each a base fragment. In this case, the function l_F maps each node to the base fragment itself ($l_F : v \mapsto \{F\}$). Other information fragments are compo-

Table 2: Sample Node Domains, Types & Properties

Domain	Types	Properties
source code (SC)	class, method, field	identifier (id), referenced elements
work items (WI)	defect, work item, plan item	id, creator, owner, linked change sets, linked comments
change sets (CHS)	change set	id, author, changed elements
teams (T)	team, team member	id, name
comments (CO)	comment	id, text, author
web/wiki (WW)	web page	id, text, last updated

sitions of several base fragments. In this case, the function l_F maps each node to the set of base fragments it is part of ($l_F : v \mapsto \{F_1, ..., F_n\}$ with $F_i = (V_i, E_i)$ and $v \in V_i$ for $i = 1, ..n$).

4.3 Composition Operators

Formally, a composition operator $\otimes : F_1 \times F_2 \mapsto F^{'}$ takes two information fragments $F_1 = (V_1, E_1, l_{F_1})$ and $F_2 = (V_2, E_2, l_{F_2})$ and creates a new information fragment $F' = (V', E', l_{F'})$. The new information fragment consists of the union of nodes ($V' = V_1 \cup V_2$). $l_{F'}$ maps each node v to the union of $l_{F_1}(v)$ and $l_{F_2}(v)$. E' is the union of E_1 and E_2 and a set of newly created directed, labeled edges E^* between two nodes, one from V_1 and one from V_2. The creation of new edges is based on node properties and is unique to the specific composition operator being used. Each composition operator is commutative. Thus, the order of the input fragments does not matter. This constraint is important for the separation of composition and presentation.

Based on the questions derived from the interviews, we determined that only two generic composition operators are needed to support answering the questions: an identifier matching and a text based matching composition operator.

ID Matching.

This composition operator, denoted by $\otimes_{id}$, creates a new edge when the identifier property (id) of one node exactly matches a property of another node. For example, the id of a team member node matches the creator property of a work item node. The label of the newly created edge is determined by the name of the property that is not the id. In our example, the newly created edge between the team member and the defect is labeled creator.

Text Matching.

The text matching operator, denoted by $\otimes_t$, creates new edges when there is a textual match between the identifier property of one node with a textual property in another node. The similarity of the identifier on the one side and the textual match on the other side can be based on some text matching measure and a threshold for similarity can be used. Consider a team member and a comment node. If the identifier property of the team member has a textual match in the text property of the comment with a high enough similarity, a new edge will be created. This new edge between the team member and the comment has properties that describe the match, such as the similarity value and the location of the match in the comment.

4.4 Presentation

Our interviews identified variations in the ways developers would like a composed information fragment to be presented. To support this variety, our approach provides a *projection* function that transforms an information fragment (a graph) into a set of trees. Specifically, given an information fragment $F = (V, E, l_F)$ composed of base fragments at the lowest level, and an ordering $(bf_1, ..., bf_n)$ of these base fragments, a projection denoted by $\phi_{(bf_1,...,bf_n)}$ creates a set of trees, TS. All tree roots are nodes of base fragment bf_1 for which a path to nodes of base fragment bf_n exists, so that the nodes on the path follow the given order. Formally, for each path $(v_1, ..., v_n)$ through a created tree $T = (V_T, E_T)$ with $T \in TS$, $V_T \subseteq V$ and $E_T \subseteq E$ the following holds: $v_i \in V_T$ for $i = 1, ..., n$, $(v_i, v_{i+1}) \in E_T$ for $i = 1, ..., n-1$, $bf_i \in l_F(v_i)$ for $i = 1, ..., n-1$ and there is no other tree $T^* \in TS$ that contains the path. In our example, $\phi_{(t,wi)}$ creates two trees (shown in Figure 1(d)) that represent all possible paths of length two from a node of the team fragment to a node of the work item fragment.

Sometimes, a developer needs to count nodes on a particular level in the presentation for a summary. For instance, several questions are about the relative occurrences of composed information, such as "What's the most popular class" (Question 23 in Table 1). Therefore, our model provides a *counting* function, denoted by $\sigma_{(levelOf(bf))}$, that counts all nodes on the level of a base fragment bf in the set of trees. For example, to find out how many work items each team member worked on in our example, a developer can count the nodes of the team fragment level ($\sigma_{(levelOf(t))}$) in Figure 1(e). This tells him that Julie and David each occur twice in the trees and thus worked on two work items. For simplicity, we shortly denote the counting function as $\sigma_{(bf)}$.

5. APPLYING THE MODEL

We have applied our model to answer the 78 questions determined by interviewing developers. Table 1 shows the domains and composition operators needed to answer each question as it was intended by the developers who stated the question. Space does not allow us to show the use of the model to answer each question. Instead, we focus on showing how the model can be used to answer 5 of the 78 questions selected to present different properties of the model.

What have people been working on? (6).

As an answer, the developer who stated this question wanted to see the list of his team members with work items on which they have been active beneath the relevant team member. Applying our model, we first need access to the information fragments of interest:

T_1 the team members of the developer,
WI_1 work items resolved recently and in progress.

We then apply the composition operator that matches identifiers ($\otimes_{id}$). The composition of these fragments can be expressed as

$T_1 \otimes_{id} WI_1$.

As the composition is commutative, the order in which the composition takes place does not matter.

To display the composed information as desired by the developer we apply a suitable projection (i.e., team first and then work items)

$\phi_{(T_1,WI_1)}$.

Overall, the answer to the question is

$\phi_{(T_1,WI_1)}(T_1 \otimes_{id} WI_1)$.

Who is working on what? (1).

This question is similar to the one above, but the developer who stated it, wanted to also see the changes made to the code. The additional information fragments are:

CHS_1 change sets recently delivered,
SC_1 source code of the project.

To reach the answer with our model, we use:

$\phi_{(T_1,WI_1,CHS_1,SC_1)}(SC_1 \otimes_{id} CHS_1 \otimes_{id} WI_1 \otimes_{id} T_1)$.

What classes has my team been working on? (11).

Answering this question requires the same information fragments as question (1). However, the developer's focus for this question was on the code and he wanted to see the work items and team members in the context of his current project. Compared to question (1), the answer differs only in the order of the projection:

$\phi_{(SC_1,CHS_1,WI_1,T_1)}(SC_1 \otimes_{id} CHS_1 \otimes_{id} WI_1 \otimes_{id} T_1)$.

Who owns/modified this piece of code most? (19).

Answering this question requires three information fragments:

T_2 all team members on the project,
CHS_2 all change sets for the last couple of months, and
SC_2 source code of interest.

To find out who made most changes over the last couple of months, the composed information is projected and then the occurrences of the team members are counted:

$\sigma_{(T_2)}(\phi_{(SC_2,CHS_2,T_2)}(T_2 \otimes_{id} CHS_2 \otimes_{id} SC_2))$.

Which conversations in work items have I been mentioned? (46).

The developer wanted to see the comments he is mentioned in ordered by work items.

T_3 the developer,
CO_3 comments recently created,
WI_3 work items of interest.

As textual matches of the developer in the comments are of interest, the team fragment is composed using the text matching composition operator ($\otimes_t$). Overall, with work items first and comments second, this results in:

$\phi_{(WI_3,CO_3,T_3)}(T_3 \otimes_t CO_3 \otimes_{id} WI_3)$.

6. PROTOTYPE

We have implemented a prototype that supports our model. Our prototype extends the IBM Rational Team Concert (RTC)[4], a team collaboration platform on top of the Eclipse IDE[5].

[4] `jazz.net`, verified 03/09/09
[5] `www.eclipse.org`, verified 03/09/09

6.1 Information Fragments

Our prototype supports four domains of information: work items, source code (Java packages, classes, methods and fields), change sets and teams (teams and team members). We enhanced existing views for these four domains in RTC to support the creation of an information fragment from elements selected in the view. We did not use this functionality in the evaluation we performed (Section 7); the next phase of our research will focus on better support for selecting fragments.

6.2 Composition

With the prototype, a developer can compose information fragments by adding them to the *composed viewer*. Figure 2(a) shows the answer to "Who is working on what?" (Question 1 in Table 1) in the composed viewer. The answer was created by adding the team, work item, change set and source code fragments described in Section 5 to the viewer. The composition operator is applied automatically when information is placed into the composed viewer. Currently, the prototype supports only the identifier matching composition operator. As discussed in Section 8, the text matching composition operator can also be applied automatically.

6.3 Presentation

The prototype supports projection and counting of our model as well as a feature for hiding certain information. Figure 2(a) displays the answer to "Who is working on what?" as described in Section 5. The projection order of the base fragments is represented by the icons in the *fragment bar* in the upper right corner of the view (see the highlighted rectangle in Figure 2(a)). If the developer is interested in a code-centric interpretation of the same question, he only has to change the order of the original base fragments by dragging and dropping the icons in the fragment bar into the appropriate order. This changes the projection; the underlying composed information fragment does not change. The resulting view is shown in Figure 2(b).

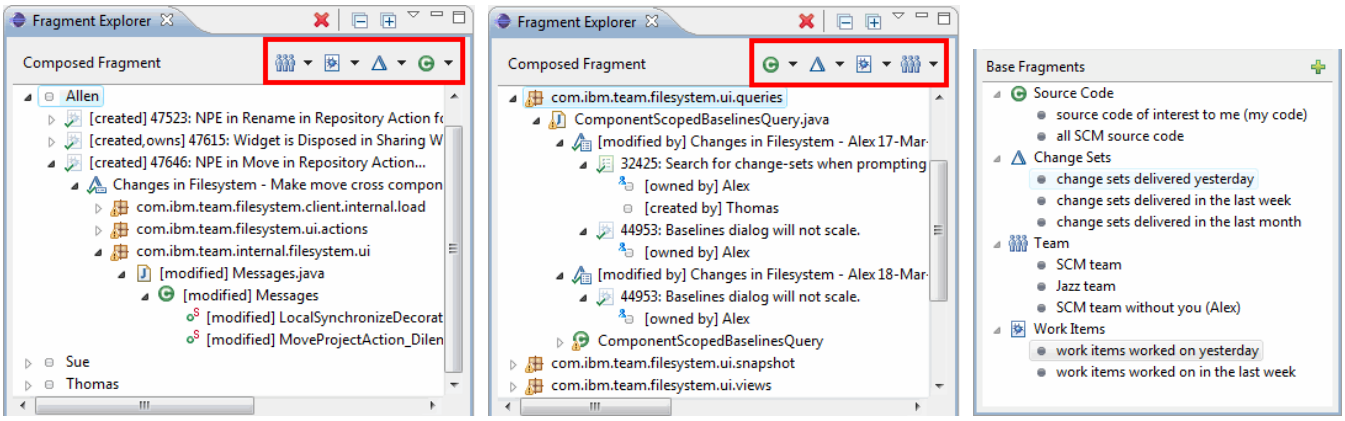
Answering a question such as which of the team members made most changes requires a count. A developer can perform this count using the drop down menu of the teams icon in the fragment bar. The count functionality counts the occurrences of nodes of a particular base fragment, in this case the team fragment, presented in the view. The developer can also count the children of a specific element. For example, he can count the team members that changed a specific Java package by right-clicking on the Java package in the view and selecting the "Count Team Members" action in the context menu.

As an additional feature for eliding information in the view, the prototype supports a hide action that allows the developer to hide nodes of a certain level. This functionality is again accessible through the drop down menus of each icon in the fragment bar.

With the above mentioned functionality our prototype supports 51 of the 78 questions mentioned in Section 3.

7. EVALUATION

To be useful, a developer must be able to easily apply the information fragment model to answer questions of interest about the software development. We considered two research questions to gain evidence about this statement:



(a) "Who is working on what?" (b) Code Centric Presentation (c) Base Fragments

Figure 2: Views of the Prototype

(1) Can developers use the information fragment model to answer questions that previous developers have posed?

(2) Can developers use the model effectively without requiring a detailed understanding of how the model works?

We conducted a study in which 18 developers used our prototype tool that supports the model to answer eight questions selected from those described in Table 1. Our study setup was an embedded, multiple-case, replication design. We chose not to perform a comparative study as the only available approach is to follow links in a development environment, an obviously time-consuming approach.

7.1 Subjects

We originally recruited 21 developers from two different locations of a multi-national company and four different teams. Seven of these 21 developers also participated in the earlier interviews. To be eligible to participate in the study, a developer had to use the RTC client in his daily work. To solicit participation, we randomly asked people at the two locations. We report on 18 of the participants: ten from one location (S1-S10) and eight from the other (T1-T8). The remaining three individuals had difficulty with the experimental situation. The roles of the 18 developers ranged from junior developer to team lead and also included one student. The professional experience ranged from 7 months to 23 years. Three of the developers (T6,T7,T8) had never used the source control mechanism in our study before.

7.2 Study Method

We selected a set of eight questions (Q1-Q8) from the 78 original questions for our study. We chose these eight to cover domains of interest as can be seen from the corresponding questions in Table 1 (second column in Table 3 shows the number of the original question). Each question was made more specific for two reasons: 1) to reduce the range of interpretations of the questions so that we could compare the approaches of participants to answering the questions and 2) to match the data available in our study setup. For example, we adapted the question

> (11) "What classes has my team been working on?"

to

> (Q5) Yesterday, on which classes (of the SCM code) have Alex and Allen on the SCM team been working on and why? For each developer, name one class and the reason for the change.

These changes to the question reflect the desire of the developer stating question 11 to see the reason for the changes in terms of work items and reflect the specification of the scope of the team, code and timespan of interest.

We narrowed Q5 to ask about one class for each developer to allow us to determine when a participant's given answer was correct. For Q5, we considered an answer correct if the participant mentioned, for each of Alex and Allen, a class which the respective developer changed yesterday and the work item requiring that change.

The version of the prototype used in the study was adapted with a view to show ten predefined base fragments, such as "all SCM source code" or "work items worked on yesterday" (see Figure 2(c)). We predefined these base fragments to focus the study on the core parts of the model: the composition and projection of information fragments. The data used for these base fragments was prepared from the history of the RTC development from a year ago, with developer names changed.

At the start of each study session, a participant worked through a 10-15 minute, two-page tutorial about the prototype's features: composition, reordering, counting and hiding. The examples used in this tutorial were straightforward, such as relying on the well-known fact that change sets contain links to their authors and, as a result, can be composed with team information. After finishing the tutorial, a participant was given time to read the eight questions (Q1-Q8) and ask clarification questions. A participant had the choice to answer these eight questions in whichever order he preferred, but was told that the questions were ordered from easier to more difficult ones.

Participants were then given time to work on the questions. If a participant spent five minutes on a question and was still not close to an answer we provided a hint. This situation only occurred for Q5 and Q8. For Q5, two participants required a hint about ordering; one participant who had not used the source control system needed a hint about

Table 3: Developer's Results

Question (orig. dev. question)	Time (in minutes) and Success per Question for each Developer ✓ = success, ■ = failure, ⋆ = hint given																		
	S1	*S2*	*S3*	*S4*	*S5*	*S6*	*S7*	*S8*	*S9*	*S10*	*T1*	*T2*	*T3*	*T4*	*T5*	*T6*	*T7*	*T8*	Mean
Q1 (21)	1.1✓	0.7✓	2.2✓	1.8✓	1.9✓	1.1✓	3.2✓	1.4✓	1.1✓	0.6✓	1.1✓	0.6✓	1.5✓	1.9✓	0.6✓	2.6✓	5.7✓	1.1✓	1.7
Q2 (15)	2.9✓	0.6✓	1.0✓	2.1✓	1.2✓	1.7✓	2.1✓	0.7✓	1.3✓	0.8✓	5.1✓	0.7✓	1.6✓	2.5✓	0.5✓	2.5✓	3.9✓	4.9✓	2.0
Q3 (14)	2.0✓	1.5✓	2.2✓	2.1✓	1.3✓	5.9✓	1.0✓	3.0✓	2.3✓	1.5✓	0.9✓	1.7✓	4.0✓	1.9✓	1.0✓	1.5✓	6.6✓	5.4✓	2.5
Q4 (6)	1.1✓	0.8✓	0.6✓	0.4✓	0.8✓	0.7✓	0.8✓	1.0✓	1.3✓	1.0✓	0.8✓	0.8✓	1.7✓	2.1✓	1.0✓	1.7✓	1.3✓	0.4✓	1.0
Q5 (11)	1.5✓	5.6✓	3.4✓	*4.2*⋆	6.1✓	3.3✓	5.8✓	1.8✓	5.0✓	1.7✓	7.2✓	*6.8*⋆	7.2✓	5.5✓	2.8✓	7.0✓	*6.5*⋆	1.4✓	4.4
Q6 (23)	1.0✓	0.8✓	0.9✓	0.7✓	0.7✓	1.8✓	3.0✓	0.6✓	5.0✓	1.2✓	1.7✓	0.6✓	2.9✓	1.6✓	0.8✓	1.5✓	1.6✓	3.5✓	1.7
Q7 (19)	2.7✓	2.6✓	3.2✓	2.6✓	4.9✓	3.4✓	1.3✓	1.5✓	2.2✓	2.0✓	2.3✓	3.5✓	2.9✓	2.5✓	1.8✓	5.2✓	1.9✓	1.6✓	2.7
Q8 (26)	1.7✓	2.1✓	4.3✓	3.1✓	– ■	5.3✓	*8.1*⋆	1.2✓	3.3✓	2.0✓	*6.5*⋆	– ■	5.4✓	5.3✓	2.6✓	*7.8*⋆	– ■	2.7✓	3.3
Mean	1.8	1.8	2.2	1.8	2.4	2.9	2.5	1.4	2.7	1.3	2.7	1.3	3.4	2.9	1.4	3.1	3.5	2.6	**2.3**

work items containing links to change sets. Six participants for Q8 were given a hint about what the question meant. If a participant was not done with a question after ten minutes we stopped him and asked him to move to another question.

After a participant finished all eight questions, we interviewed the participant about the experience of using the tool.

7.3 Data Analysis

For each participant, we captured screen videos as the participant worked and took written notes. From our notes, we determined, for each question, the worst case time that it took the participant to get to the correct answer based on our interpretation of correctness. Only for Q5 was there a distinct difference between our notion of correctness and the participant's notion of correctness. For Q5, eight participants (S3,S6,S7,S10,T3,T4,T6,T8) interpreted the reason for a change as the one-line description of the change set, and did not consider work items. When directed by the experimenter to consider work items, the participants continued working on the question, taking, on average, an extra three minutes to get to the correct answer. We used only these worst case times for our analysis.

We used the video to analyze the interaction of a participant with the features of the tool.

7.4 Results

To evaluate our first research question "Can developers use the information fragment model to answer questions that previous developers have posed?", we consider how many of the eight questions participants were able to answer correctly. If our model is usable, we expect participants to succeed in answering most questions within the ten minutes time limit. For our model to improve on a query approach, participants should succeed in answering most questions in five minutes as studies of SQL have shown that users, after substantial training (1.5 hours), can write queries that join elements from two domains in a mean time of 5.1 minutes [1].

Table 3 shows, for each question in the study, the time and success per developer, the mean time needed for each question in the last column and the mean time per question for each developer in the bottom-most row. The data in the table shows that in 135 of the 144 cases (94%), participants answered the questions successfully without any hint and with a mean of 2.3 minutes per question; in computing the mean, we did not include cases (shown in italics in Table 3) in which a hint was given or the ten minutes time limit was exceeded. We consider that this data shows strong support for the usefulness of our model.

In only 23 cases (16%), participants required more than five minutes to answer the question and a hint was given in six cases. In one case (S4), a hint was given before the 5 minutes as the subject was under time pressure. Two of the three subjects (T6-T8) that did not have any prior knowledge of the source control mechanism in use had the highest mean time for answering a question. However, even these participants successfully completed almost all questions with a mean time of less than five minutes per question. These results support the statement that developers can use the information fragment model to answer questions they face.

To evaluate our second research question "Can developers use the model effectively without requiring a detailed understanding of how the model works?", we consider how much information we needed to provide a participant while working on a question and how a participant used the model to answer questions. In terms of information provided to participants, we gave participants only ten to fifteen minutes of training on the model; in comparison, in a study of SQL, Chan and colleagues trained participants for an hour and a half [1]. Despite this low amount of training, for 135 cases (94%) this training was sufficient for the participants to use the model to answer questions correctly. In only 9 cases (6%), an additional hint was given by the experimenter. Considered alongside a mean time of only 2.3 minutes per question for successful answers, this data provides support for the statement that our model can be used effectively without understanding the details of the model.

We also considered how a participant used the model to answer questions. By analyzing the screen captures of the participants working, we found that the participants reordered the information shown in the composition view more often (277 times) than the participants restarted the composition (213 times). Participants stated in follow-up interviews that the ability to change the projection of the composition through reordering was "definitely helpful" as "you can really do it in two steps"(S10), that "I throw everything in it [the view], reorder it [...] and see if it seems reasonable"(S2), and even that it was "great" (T8). The higher number of reorderings compared to restarts and comments of the participants provide further evidence that the par-

ticipants understood enough features of the model to use it effectively without extensive training.

When asked explicitly, all developers stated that, if available, they can imagine and would use the prototype. Some developers even stated without us asking that it was "pretty nifty"(S4), "pretty cool"(S3), "cool"(S7,S6), a "neat tool"(T2), "really really cool"(T8) and that "it is answering questions that I don't think we can answer right now"(T1).

7.5 Threats

We predefined the information fragments available for use in the study. As these information fragments were tailored towards the questions asked, participants may have had to spend less time than otherwise to answer a question of interest. On the other hand, these predefined fragments also made it more difficult for a participant to restrict the information shown to what he desired seeing.

8. DISCUSSION

Automatic Composition.

Our prototype supports the answering of 51 of the 78 questions with the use of only the id matching operator. Since our study used eight of these 51 questions, a developer did not have to specify an operator, rather it was chosen automatically. For the 27 remaining questions that also involve the text matching operator, it can be shown that the composition operator can again be chosen automatically. Whether or not future operators that are added for additional questions can also be chosen automatically is an open question.

Information Fragment Selection.

We have focused our attention to date on the composition and presentation of information fragments, using simple mechanisms or pre-defined fragments in the prototype tool we have developed. A future research question is how best to support a developer in selecting fragments of interest. It may be that it is possible to predefine fragments for such a large number of questions to make sophisticated selection mechanisms unnecessary. We leave this investigation to future work.

9. CONCLUSION

Software development involves working with many different kinds of artifacts and coordinating work with others. These activities lead to questions about the development that span multiple sources of information, such as bug reports, source code, team membership and others. In this paper, we have introduced the information fragment model that supports the composition of information from multiple sources and that supports the presentation of composed information in flexible ways. We have shown that this model can support 78 questions software developers want to ask about a development project and that, for these questions, the composition can be chosen automatically by a tool supporting the model. Finally, we have demonstrated that 18 professionals, with minimal training, were able to use the model, as supported by the prototype tool, easily and effectively.

10. ACKNOWLEDGMENTS

Thanks to Annie Ying and Emerson Murphy-Hill for helpful comments. This work was supported in part by IBM and in part by NSERC. We thank all the developers who participated in the interviews and case study.

11. REFERENCES

[1] H. Chan, K. Siau, and K.-K. Wei. The effect of data model, system and task characteristics on user query performance: an empirical study. *SIGMIS Database*, pages 31–49, 1997.

[2] D. H. Chau, B. Myers, and A. Faulring. What to do when search fails: finding information by association. In *Proc. of CHI '08*, pages 999–1008, 2008.

[3] B. de Alwis and G. C. Murphy. Answering conceptual queries with ferret. In *Proc. of ICSE'08*, pages 21–30, 2008.

[4] T. Fritz and G. C. Murphy. Search, stitch, view: Easing information integration in an IDE. In *SUITE'09*, pages 9–12, 2009.

[5] E. Hajiyev, M. Verbaere, and O. de Moor. Codequest: Scalable source code queries with datalog. In *Proc. of ECOOP'06*, pages 2–27, 2006.

[6] R. Holmes and A. Begel. Deep intellisense: a tool for rehydrating evaporated information. In *Proc. of MSR'08*, pages 23–26, 2008.

[7] D. Janzen and K. D. Volder. Navigating and querying code without getting lost. In *Proc. of AOSD'03*, pages 178–187, 2003.

[8] C. Kiefer, A. Bernstein, and J. Tappolet. Mining software repositories with isparql and a software evolution ontology. In *ICSEW '07*, page 10, 2007.

[9] A. J. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *Proc. of ICSE'07*, pages 344–353, 2007.

[10] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: a study of developer work habits. In *Proc. of ICSE'06*, pages 492–501, 2006.

[11] M. A. Linton. Implementing relational views of programs. In *Proc. of SDE 1*, pages 132–140, 1984.

[12] S. Paul and A. Prakash. A query algebra for program databases. *IEEE Trans. Softw. Eng.*, pages 202–217, 1996.

[13] D. Quan, D. Huynh, and D. R. Karger. Haystack: A platform for authoring end user semantic web applications. In *Proc. of ISWC'03*, pages 738–753, 2003.

[14] J. Sillito, G. C. Murphy, and K. D. Volder. Questions programmers ask during software evolution tasks. In *Proc. of FSE'06*, pages 23–34, 2006.

[15] D. Čubranić, G. C. Murphy, J. Singer, and K. S. Booth. Hipikat: A project memory for software development. *IEEE Trans. Softw. Eng.*, pages 446–465, 2005.

[16] Y. Ye, Y. Yamamoto, and K. Nakakoji. A socio-technical framework for supporting programmers. In *Proc. of ESEC-FSE'07*, pages 351–360, 2007.