Creating Socio-Technical Patches for Information Foraging: A Requirements Traceability Case Study

Anonymous Authors

***Abstract*—Work in information foraging theory presumes that software developers have a predefined patch of information (e.g., a Java class) within which they conduct a search task. However, not all tasks have easily delineated patches. Requirements traceability, where a developer must traverse a combination of technical artifacts and social structures, is one such task. We examine requirements socio-technical graphs to describe the key relationships that a patch should encode to assist in a requirements traceability task. We then present an algorithm, based on spreading activation, which extracts a relevant set of these relationships as a patch. We test this algorithm in requirements repositories of four open-source software projects. Our results show that applying this algorithm creates useful patches with reduced superfluous information.**

***Index Terms*—Information foraging theory, Spreading activation, Requirements analysis, Traceability**

I. Introduction

If we understand how a user seeks information, then we can optimize an information environment to make that information easier to retrieve. Pirolli and Card worked to understand information-seeking by defining information foraging theory [1]. Information foraging theory describes a user's information search by equating it to nature's optimal foraging theory: in the same way that scent carries a predator to a patch where it may find its prey, a user follows cues in their environment to information patches where they might find their information.

Information foraging theory has seen many applications since Pirolli's seminal work. For example, in web search, foragers follow information scent to their patches, web pages [2], [3]. Understanding how foragers find information in web search has helped developers design the information environment of their web pages [3]. In code navigation and debugging, information foraging theory describes how developers seek to resolve a bug report by navigating from fragment to fragment of code to define and fix the problem [4]. By understanding the process of finding and fixing bugs, models can be written to assist in this process [5]. In both of these scenarios, the patch is clearly defined: in web search, a forager patch is a web page, and in debugging, a developer's patch might be a fragment of code. What happens, though, when the patch is not clearly defined?

Consider socio-technical systems, where information artifacts are connected to people. Facebook, YouTube, Twitter, GitHub, and Wikipedia all have information artifacts, like posts and code snippets, with a rich context of social interactions tying them together. A forager traverses both the artifacts and the social structures behind them in an information seeking task. Therefore, both artifacts and social structures should be considered when defining a patch, and patches are not necessarily immediately evident. Consider a user wondering how a reposted image became so popular among their friends: how could patches be defined for this forager? Photos, posts, comments, friends' connections, some combination? In this paper, we describe a method for delineating patches in such environments.

Requirements traceability is an ideal field for examining patch creation in a socio-technical environment. Requirements traceability is a socio-technical system used to describe and follow the life of a requirement by examining the trail of artifacts and people behind them, from the requirement's inception to implementation. Requirements traceability problems, as studied by Gotel and Finkelstein [6], arise when questions about the production and refinement of requirements cannot be answered. More specifically, with a traceability failure, US Food and Drug Administration might cast doubt in product safety [7]. With a traceability failure, the CEO of a prominent social media company cannot explain to Congress how a decision to withhold information from customers was made [8]. Applying information foraging to these problems could significantly increase efficiency and efficacy of these traceability tasks.

We relate requirements traceability to information foraging theory and its patches by considering requirements traceability questions. We define a requirements traceability question as a query that a project stakeholder issues *in situ* wanting to know a requirement's life. A requirements traceability question is where a user's traceability task becomes a foraging task; the question represents the user's information need, or foraging prey. If questions represent a traceability forager's prey, what represents a traceability forager's patch? Put simply, we aim to answer the research question: *where should a user search to understand their requirements traceability question?*

This paper makes two contributions by deriving a method for delineating these patches. First, by examining requirements socio-technical graphs constructed from four requirements repositories containing 111 traceability questions, we identify classes of relationships that should be considered in similar requirements traceability tasks. Second, we derive an algorithm, based on spreading activation, which combines these classes with information foraging concepts to create relevant patches where foragers can conduct their traceability tasks. The patches that our algorithm produces are as small as 5-10 nodes—a manageable quantity for a forager—representing knowledgeable users and useful information artifacts. The

method for identifying these classes and deriving this algorithm can be extended to other socio-technical tasks.

II. Background

A. Information Foraging

Information Foraging Theory (IFT) [9] simplifies, in a principled way, analyzing information-seeking tasks by providing constructs borrowed from its optimal foraging theory roots. Optimal foraging theory describes predators that pursue prey through an environment, following scent from locality to locality. The predator is always trying to optimize their task. In IFT, the information seeker pursues information through an information environment. Scent, in the information environment, is a construct that exists in the forager's mind, representing their perception of where they might find information; this perception is shaped by proximal cues—hints provided by the information environment. Information foragers follow scent from locality to locality, information patch to information patch, pursuing their prey.

The constructs provided by IFT were first used to analyze how a web user might search for information online [2], modeling scent as relatedness of a link to the forager's prey. This work eventually developed into the WUFIS (Web User Flow by Information Scent) algorithm [3]. WUFIS represents network topology as a graph, where nodes are web pages and edges are the links that a user can click to navigate from one to another. IFT's scent is represented by relatedness of the words in a webpage to the forager's information need. The algorithm then predicts where the user will navigate by applying spreading activation, an algorithm which will be later detailed. Spreading activation assigns each node with a value which represents the probability that a forager, given their current location and information need together with the scent of links connecting pages, will navigate to a specific page.

This spreading activation concept was utilized to model programmer navigation in the development of PFIS [10] and its subsequent revisions [11], [12]. PFIS built upon the spreading activation of WUFIS by applying it to the field of developer navigation [10], inferring the forager's goal [11], and creating multi-factor models with PFIS [12]. In the programmer navigation domain, WUFIS web page nodes were now code fragments, and its edges were any click-able link that would navigate a developer from one fragment to another. When inferring the forager's goal, PFIS authors introduced the concept of heterogeneity to their network: in addition to linking code fragments, the PFIS algorithm also linked code fragments to key words (e.g., those extracted from a bug report), creating a more nuanced topology. Inspired by this heterogeneous approach, we take spreading activation to the socio-technical realm.

B. Socio-Technical Networks

In order to develop a spreading activation algorithm in the socio-technical realm, we first examine work conducted in socio-technical graphs. In the Codebook [13] project, people and work artifacts were "friends" in a social network. A user might be connected to an email they sent, bug they closed, and a commit they pushed; that commit has changes in code containing classes and calls. By using a single data structure to represent these people, artifacts, and relationships, and a single algorithm (regular language reachability) to analyze this graph, Codebook could handle all the inter-team coordination problems identified in a survey responded by 110 Microsoft employees [14], including requirements traceability problems.

Codebook addresses problems by having a project personnel cast their coordination needs into regular expressions. For example, the requirements traceability question "Which program manager wrote the specification for that code?" could be addressed in Codebook with the regular expression "*Code MentionedBy WordDocument AuthoredBy Person*". This is a manual task, requiring a domain expert. In contrast, spreading activation can provide a mechanism for automated querying. We therefore adopt Codebook's underlying data structure, but instead of regular language reachability, we adapt the people-artifact graph for spreading activation.

III. Examining Requirements Socio-Technical Graphs

In order to successfully create patches in a requirements traceability environment, we must first understand the characteristics of the environment. To do this, we construct graphs of the environment following the Codebook paradigm. We then examine the types of human-human and human-artifact relationships that connect a traceability question to an identified answer; encoding these relationships can build a patch that a requirements traceability forager can explore to understand their prey—their need to understand their traceability question—better.

A. The Information Environment

The literature suggests that issue trackers are essential for open-source projects to manage requirements [15]–[19]. Although the requirements of an open-source project can originate from emails, how-to guides, and other socially lightweight sources [20], the to-be-implemented requirements "eventually end up as feature requests in an issue-tracking system" [17]. We therefore turn to the issue-tracking system Jira to understand the life of requirements.

From Jira, we select four open-source projects from the Apache software foundation [21] or the JBoss family [22].

TABLE I
Jira Projects and Characteristics

Project	Domain	Written	Initial & Latest Releases Examined	|Ques -tions|
DASH-BUILDER	data reports	Java, HTML	Aug 27, 2014 & Apr 14, 2016	31
DROOLS	business rules	Java	Nov 13, 2012 & Jul 17, 2017	57
IMMU -TANT	complexity reduction	Clojure	Mar 14, 2012 & Jun 23, 2017	18
JBTM	business process	Java, C++	Dec 5, 2005 & Jul 14, 2017	5

TABLE II
SELECT QUESTIONS WITH IDENTIFIED ANSWERS FROM DROOLS

id	issue-jira-key	body	asker	answered by	role	operations
1206	DROOLS-972	Can you please send a PR adding a new example in …	Mario Fusco	Mauricio Salatino	Creator	add pull request
1220	DROOLS-963	This should be fixed for 6.4.0.Final, does it sound possible?	Mauricio Salatino	Petr irok	Assignee	change fix version to 6.4.0
1371	DROOLS-907	As kie-group website is not prepared for rake I have …	Michael Biarnes Kiefer	Geoffrey De Smet	Creator	marked as done
1377	DROOLS-905	Thanks for the pointers [martenscs]. Is there any …	Petr irok			
1383	DROOLS-901	any news here [¯mfusco]?	David virgil naranjo	Geoffrey De Smet		provide opinion and reason
1627	DROOLS-823	Hi Geoffrey, When an error is found, what would …	Michael Biarnes Kiefer			

The four projects tackle problems in different domains with implementations written in different programming languages, as seen in Table I.

Within the chosen projects, we focus on questions and answers. As discussed, questions represent a traceability forager's information need. By finding the exact person who answered a forager's question, we can gain insight into the information environment. For each project, two researchers identified comments that were questions and identified the respective answer comments, building their answer sets individually. Some examples of the answer sets generated can be seen in Table II. The researchers reached a substantial degree of agreement (average Cohen's kappa=0.67) on requirements traceability questions over the 4 projects. Discrepancies were resolved in a joint meeting between the researchers.

In Jira, issues (typically tasks or feature requests) are our requirements. Comments are provided to the issue, and may contain questions or answers. Users create issues, are assigned to issues, submit comments, and reference other users within their comments. Contemporary approaches to requirements traceability are either artifact-based (e.g., trace retrieval) and would consider only the comments and issues [23], or are driven by social roles (e.g., contribution structures in the requirements specification production) and would consider the relationships of the people [24]. As we constructed our answer set, however, we realized the impact of the people-artifact relationships on the ability to track a requirement's life. Therefore, we consider Jira's issues, comments, *and* people in our information environment.

B. Constructing Requirements Socio-Technical Graphs

The next step to creating patches in this network is to build a requirements socio-technical graph (RSTG) so that we can inspect the topology of the network using graph theory concepts. Issues, comments, and people serve as the nodes in our network. What, then, serve as the edges between our nodes?

By manually inspecting the paths between questions and their eventual answers, we are able to define the edges in our network topology. Consider the following example: Figure 1 is a subgraph from the IMMUNANT project, showing two questions and their answers. User 2881 created Issue 73650. User 6655, the forager, commented on the issue (Comment 149789), asking User 2881 for clarification by referencing them in the comment. User 2881 commented on the issue (Comment 149790), providing the clarification. In another foraging interaction, User 2881, now the forager, commented

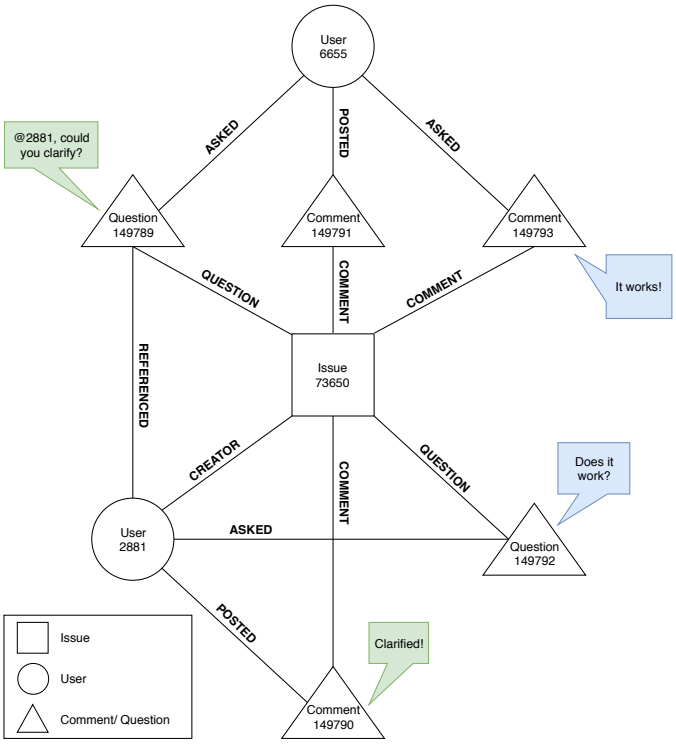

Fig. 1. Requirements Socio-Technical Graph

on the issue (Comment 149792), asking "This is now available in [incremental build—http://immutant.org/builds/2x/] 591 and newer. Can you give that a try and confirm it works for you?" User 6655 responded (Comment 149793) "I've finally found the time to test this and can confirm it works! Thanks!".

These two foraging interactions demonstrate most of the observed relationships between nodes in the datasets. These relationships are treated as the edges within our graph:

- *Creator:* Users create issues
- *Commented, Asked:* Users post comments and questions
- *Comment, Question:* Comments and questions are posted to Issues
- *Referenced:* Comments can reference other users
- *Assignee:* (Not pictured) Users can also be designated assignees on issues

While some relationships could be considered as unidirectional, e.g., a user posting a comment, the comment also serves as a bridge connecting the issue to the user, who is knowledgeable on the issue. We therefore elected to encode the network as an undirected graph.

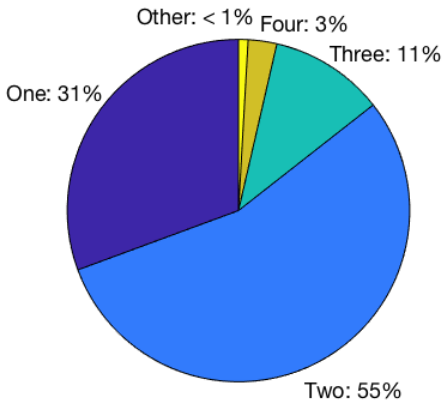

Fig. 2. Degrees of Separation between Question and Answer in Projects

By traversing the issues, comments, and users of each Jira project, and connecting each with these relationships, RSTGs like Figure 1 are constructed. An RSTG was constructed for each question, representing the state of the network at the time of the question. That way, only relationships that had existed prior to the asking of the question could be considered.

While each issue has its creator and assignee explicitly defined by Jira, and each comment has its author and issue, some additional processing was required to identify questions and references. Questions were identified by natural language recognition. In practice, a reference can either be the user's username (e.g., [˜ge0ffrey]) or by first name (e.g., "Geoffrey," or "Geoffrey:"). With regular expressions, usernames were identified in the body of a comment and connected the comment to the user. However, first-name references had to be manually recognized and connected to their usernames.

C. Properties of Requirements Socio-Technical Graphs in Information Foraging

We now tie our RSTGs to information foraging theory. When a forager asks a question, like the ones in Figure 1, where do they find their answer? What path might the traceability forager typically follow to the user who will provide the answer, gathering information on their prey along the way?

By analyzing the paths connecting question nodes and answer nodes, we identified recurring patterns connecting traceability questions with answers. The patterns are organized by degrees of socio-technical separation, which we define as the minimum number of edges to be traversed between two nodes. Figure 2 displays the frequency of degrees separating questions and answers for our 111 questions.

1) One or Two Degrees of Separation: Figure 2 shows that more than three-quarters of answers are within two degrees of the question. These represent simple foraging tasks. By the time a forager posts a question, they may be relatively familiar with an issue. The very fact that they chose a specific issue to comment on demonstrates that they expect their answer to be near the issue. A well-informed forager will often reference the exact user they expect to know the answer. These kinds of relationships fall within this category.

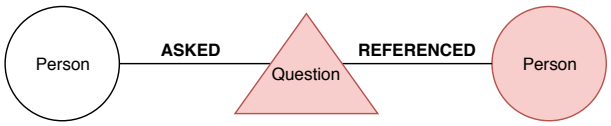

Fig. 3. Referenced (1 Degree)

Within one degree of the question, two types of answers were observed. The first is when the forager answers their own query. Because the forager, in the RSTG, is directly connected to their comment, this is one degree of separation. The second is when a user is referenced within a comment. As shown in Figure 3, when a user is referenced in a comment, they are directly connected to the comment. Comment 70842, from the DROOLS project, is a good example of this: "What should the URL look like in your opinion [˜tirelli]?" asks User 4079. User tirelli answers. A vast majority of the 31% of answers one degree away from the question came from direct references.

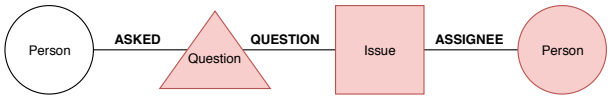

Fig. 4. Creator or Assignee (2 Degrees)

Within two degrees of separation, we observed two more types of answers: when a forager asks a question on an issue, either the issue creator or issue assignee responds. Figure 4 shows this interaction. When a forager posts a question to an issue without a reference, the creator, with their knowledge of what the issue is, or the assignee, with knowledge of what is being done to solve the issue, can answer. In Comment 353748, from the JBTM project, User 133 posts to an issue: "Status update please?" User 3619, assignee, provides an answer.

2) Collaborators and Contributors—Three or Four Degrees of Separation: More challenging foraging occurs when the user with the answer is three or more degrees of separation away from the question. With each degree, the number of information artifacts and other users a forager must traverse increases exponentially. Identifying patterns and presenting useful patches to the forager in this class of foraging presents far greater potential. We present a few of the most frequent types of interactions that occur within this radius.

Most answers in our projects within three degrees of separation were of the type shown in Figure 5: Comment–Person–Issue–Person. The question connects to a person—either the traceability forager or a referenced user—and the answer comes from a *Frequent Collaborator*. While neither the forager themselves nor a referenced user has an answer, someone they collaborate with does. The Collaborator is connected with the user or the referenced by one or more comments (where the Collaborator is directly referenced). We notice

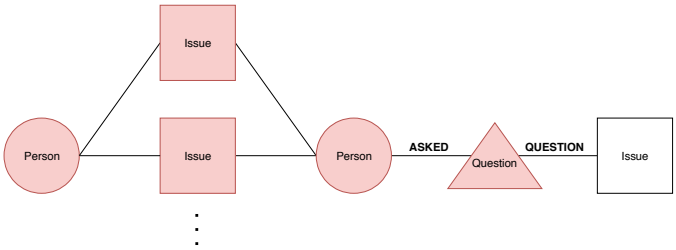

Fig. 5. Frequent Collaborator of Asker or Referenced User (3 Degrees)

that Collaborators are often highly-central users in a project, whose Frequent Collaborator status arises from their frequent contributions to projects. In other words, they are Frequent Collaborators because they are frequent users in general.

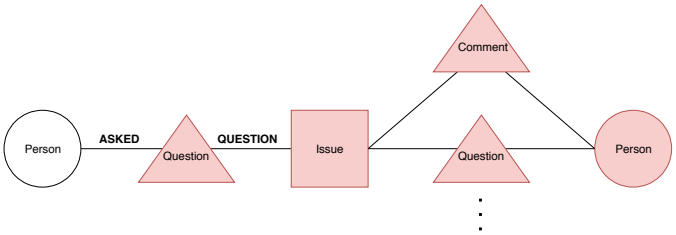

Fig. 6. Frequent Contributor to Issue (3 Degrees)

The second type of answer within three degrees of separation is that shown in Figure 6: Comment–Issue–Comment–Person. We call this type of answer a *Frequent Contributor*. This user does not directly know the forager, nor are they creator or assignee of the issue. However, the Contributor has commented and asked on a given issue one or more times. Again, Frequent Contributors tend to be highly-active, and therefore highly-central users. Figure 2 shows that Frequent Collaborators and Frequent Contributors represent 11% of answers in our dataset.

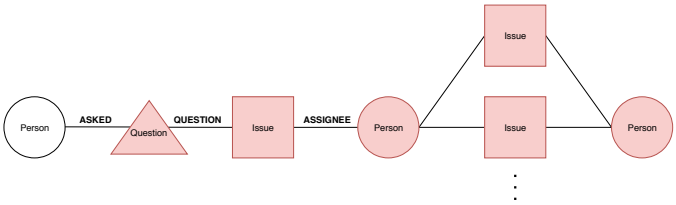

Fig. 7. Frequent Collaborator of Creator or Assignee (4 Degrees)

Within four degrees, we see another variant of The Collaborator: a *Frequent Collaborator of the Creator or Assignee* (Figure 7). This type of relationship is signified by the pattern Comment–Issue–Person–Issue–Person. The creator or assignee of the issue has collaborated with the answerer on other issues before, either as creator or assignee on those issues. While the creator or assignee of the primary issue being considered may not have the answer, someone they often work with may. Figure 2 shows that these kinds of interactions represent 3% of our dataset.

We hypothesize that, within three or four degrees of separation, several other types of collaborators could be observed. Frequent Collaborators of the Asker could manifest by Comment–Person–Comment–Person, rather than Comment–Person–Issue–Person as we observed. Within four degrees, we observed Frequent Collaborators of Creators or Assignees; a Frequent Collaborator of the Asker could also show up within four degrees (Comment–Person–Issue–Comment–Person, or Comment–Person–Comment–Issue–Person).

3) Five or More Degrees of Separation: In our sample set of 111 questions, there were no instances of five or more degrees of separation. Within this class, though, we hypothesize that more interesting variations of Frequent Collaborators and Frequent Contributors would arise; if one user does not know the answer, someone they know will. However, not having observed this class, we do not anticipate that it will be a commonplace sight.

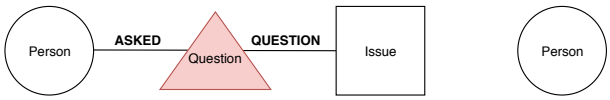

Fig. 8. Answer disconnected from network

4) Disconnected Users: The final pattern observed was one instance of an answer unconnected from the graph of the project (Figure 8). At the time the question was asked, the user who will eventually answer the forager's traceability question was not yet connected to the project by the relationships we chose to express as edges. When that user finally did connect to the the network, they were four degrees of separation away (Comment–Person–Comment–Issue–Person), appearing as the assignee to an issue where the asker had commented. With a larger dataset, these not-yet-connected relationships would be interesting to assess. More edge and node types could enable a connection between that user and the question.

TABLE III
NUMBER OF NODES WITHIN N DEGREES OF QUESTION

	Min	Q1	Med	Q3	Max
1 Degree	4.00	42.00	227.00	488.00	2461.00
2 Degrees	5.00	278.00	575.50	1317.00	4166.00
3 Degrees	5.00	567.00	1939.50	3177.00	7594.00
4 Degrees	5.00	729.00	2686.50	4163.00	7953.00

It appears, from these classes, that a substantial amount of traceability foraging takes place in four-or-less degrees of separation. Seeking to apply information foraging to traceability, one could simply present all nodes within three-to-four degrees of the question as a patch where the forager might seek to understand their question. This would satisfy our requirement of encoding frequently-traversed foraging paths into a patch. However, as shown in Table III, these patches are extremely large. With some notion of relatedness, however, these patches could be made smaller without losing relevant information.

IV. Creating Socio-Technical Patches for Information Foraging

In order to create smaller patches which still contain the described relationships, we turn to the foraging concept of scent. Following the example of WUFIS and PFIS, we define the foraging concept *Information Scent* as the "inferred relatedness of a cue to the prey, as measured by amount of activation from a spreading activation algorithm." In WUFIS and PFIS, relatedness between two nodes was determined by a function of textual similarity and encoded into the weights of the edge connecting those two nodes. The application of spreading activation introduced a notion of proximity; each node's activation is representative of its textual similarity and proximity to the starting point—the information need. Our challenge is taking the work of WUFIS and PFIS and defining these concepts in our socio-technical context. Once we have a measure of relatedness in traceability foraging, we can create patches.

A. Spreading Activation over Requirements Socio-Technical Graphs

If a forager posts a question on an issue, the user who will provide the answer will likely require knowledge on that issue. From this postulate, we define our notion of "relatedness" between two nodes: the degree of a user's relatedness to an artifact is proportional to their knowledge on the artifact. This serves as an analogue to PFIS's textual similarity. The user's knowledge on an artifact is encoded as the weight on the edge connecting them. Note that, to support our implementation of spreading activation, a lower weight represents a more powerful connection.

- *Comment to Issue, Question to Issue:* **weight = 1**. This is the only type of artifact-to-artifact edge we defined in our RSTGs. If a comment or question is posted to an issue, it is directly a part of the traceability history of that issue. Therefore, its connection to the issue should be extremely strong.
- *Creator to Issue, Assignee to Issue:* **weight = 2**. As discussed, the creator and assignee have a high degree of knowledge on a given issue. In relationships connecting question to answer, the assignee or creator frequently answered questions if the forager did not directly reference a user first (Figure 2: 55% of questions). However, an issue can develop without the supervision of the creator or assignee. Therefore, the connection of this relationship is lower than that of an issue to a comment.
- *Comment to Referenced:* **weight = 2**. The user who wrote the comment has determined that the referenced user should have strong knowledge on the comment. Figure 2 shows that 31% of questions were answered by a referenced user; this relationship is therefore prioritized with a strong connection.
- *Comment to User:* **weight = 3**. While the connection of a comment to the issue is strong, the user is not guaranteed to have knowledge on the issue to the same degree as Creator or Assignee. Therefore, we set the user's connection to the issue lower than that of a creator or assignee.
- *Question to User:* **weight = 4**. Questions, like comments, do not guarantee knowledge on the issue. If anything, a user asking a question is least-qualified of the users discussed to provide an answer. Therefore, we give this relationship the weakest connection.

With weight defining the relatedness between two given nodes, now a given node's relatedness to the forager's information need can be determined. The classes discussed in the previous section demonstrated that answers fell within four degrees of separation, and that answers were likely to come from Frequent Collaborators. These concepts can be encoded by adapting spreading activation.

Algorithm 1 Spreading Activation over an RSTG

Spreading Activation

Input: Graph, Source Node
Output: Graph with Activated Nodes

1: $Decay \leftarrow 0.1$
2: $source.activation \leftarrow 1$
3: **for** $node$ in Breadth-First Traversal from $source$ **do**
4: **for** $preNode$ in Neighbors of $node$ **do**
5: **if** $preNode$ has been traversed **then**
6: $weight \leftarrow edge(preNode, node)$
7: $newActivation \leftarrow preNode.activation * (1 - weight * decay)$
8: $node \leftarrow Activate(node, newActivation)$
9: **end if**
10: **end for**
11: **end for**

Co-Routine: Activate

Input: Node, New Activation
Output: Activated Node

1: $frequencyReward \leftarrow 0.01$
2: **if** $node$ has activation **then**
3: $maxAct \leftarrow max(node.activation, newAct)$
4: $minAct \leftarrow min(node.activation, newAct)$
5: $node.activation \leftarrow maxAct + ((1 - maxAct) * minAct * frequencyReward)$
6: **else**
7: $node.activation \leftarrow newActivation$
8: **end if**

As shown in Algorithm 1, our variant of spreading activation starts at the question-node. The question's activation is set to 1. Then, surrounding nodes are traversed (Algorithm 1, Line 3), and activation is spread from their predecessors (Algorithm 1, Lines 4-8). Spreading activation traditionally has each node firing to its successors; our predecessor variant exhibits greater decay while still producing useful networks. Our exact mechanism of spreading activation is shown in Algorithm 1, Line 7, and the Algorithm 1 Co-Routine.

Line 7 shows how exactly weight is factored in. A lower weight implies a higher degree of relatedness. If activation is a measure of relatedness, lower weights should result in higher activations. Therefore, the lower the weight, the smaller the effect of the decay, and the higher the resulting activation.

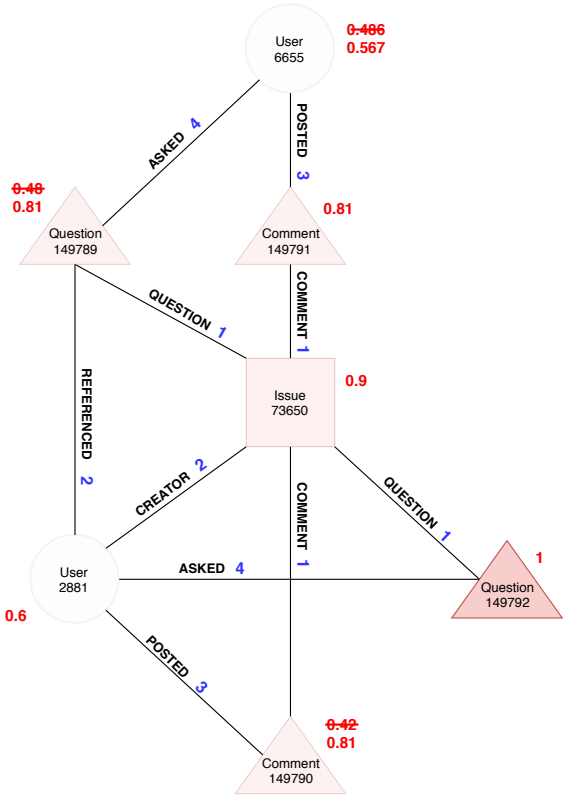

Fig. 9. Spreading Activation Applied to an RSTG (Without Frequency Bonus)

The co-routine is divided into two branches by a conditional structure (Algorithm 1, Co-Routine, Line 2). If a node has no activation yet, the activation is simply spread to the new node. However, if a node already has activation, which activation is considered? Following the example of PFIS2 [11], the higher activation value is spread. However, in order to incentivize frequency (in order to promote the Frequent Collaborator and Frequent Contributor patterns), if a node already has activation (i.e. the node has an existing relationship to the question), a percentage of that existing relationship is added to the new activation. Currently, this frequency reward is set to 1% (Algorithm 1, Co-Routine, Line 1); otherwise, highly-central figures gained disproportionately high activations, and therefore activations did not suitably diminish throughout the graph. Another method for reinforcing frequent collaboration could be a subject for future work.

Figure 9 provides an example of this variant of spreading activation being applied to the RSTG from Figure 1. Each of the relationships is assigned its weight, and the activation of Question 149792 is set to 1. Then, User 2881 and Issue 73650 are visited and activation from the question is spread. Comment 149790 and Question 149789 had activation spread from both User 2881 and Issue 73650; the higher activation was retained.

B. Delineating Patches within Spreading Activation RSTGs

With spreading activation encoding relatedness with the traceability forager's information need, patches can be created. Recall, from WUFIS and PFIS, that activation is a measure of information scent. By grouping together nodes with high activation, a patch with nodes related by the relationships described previously will be created. To do this, though, a threshold of "high" activation must be defined.

Earlier, creating patches by simply enclosing all nodes within four degrees of separation was proposed, because all answers in our dataset fell within four degrees. We now consider those answers' activations. Examining graphs of the classes discussed earlier, with spreading activation completed, reveals that a forager setting the cutoff at 0.72 would include 84% of results. Setting the cutoff at 0.45 would include all results. Examining earlier classes revealed that many Frequent Contributors and their related information artifacts fall above 0.56; many Frequent Collaborators fall around 0.50.

V. Results and Analysis

A. Quantitative Evidence

The descriptive statistics in Table IV suggest that delineating patches by activation creates smaller patches than by degrees of separation (Table III). Statistically comparing 4 Degrees and Activation ≥ 0.45, we conclude that the two sets are non-identical (t = 10.901, p-value < 0.01). A similar test for Activation ≥ 0.45 and ≥ 0.72 reaches the same conclusion (t = 9.6481 p-value ≥ 0.01). In other words, each cutoff has significantly smaller patches than the previous.

TABLE IV
Patch Size With Cutoff

	Min	Q1	Med	Q3	Max	Answer
4 Degrees	5	729	2686	4163	7953	100%
A $\geq$ 0.45	5	273	649	1860	4834	100%
A $\geq$ 0.72	2	4	6	10	24	84%

While patch sizes at cutoff Activation ≥ 0.45 are still too big for timely foraging, patch sizes at ≥ 0.72 are substantially more reasonable. That being said, ≥ 0.72 patches frequently do not include the answer node for three or four degree of separation relationships. However, within these patches, we believe that foragers would still find information relevant to their information need.

B. A Practical Example

Figure 9 serves as a practical example of both the mechanism of the algorithm and a tradeoff to consider. Figure 9 is a network with a cutoff set to ≥ 0.56—a cutoff chosen for its inclusion of many Frequent Contributors. Indeed, it was a Frequent Contributor that answered User 2881's question; User 6655 had commented twice on Issue 73650 already. While the question ("Does it work?") was a pointed request, asking for a specific piece of information, the patch generated by the

request yields not only the user who will answer the request, but also related traceability information.

Figure 9 also demonstrates a limitation of the algorithm in its current state. Other implementations of spreading activation begin from one or more nodes; we could have started the activation from the question *and* the asking user. We chose to include only the question node, as to avoid superfluous information from the asking user's connections. In this case, though, had we included User 2881 as an initial node for activation, the algorithm would have assigned higher activations to the direct collaboration between User 6655 and User 2881 (User 2881–Question 149789–User 6655). This collaboration was key to the traceability history of Issue 73650.

VI. Implications

Piorkowski and his colleagues [4] codified the fundamental challenges faced by software developers when foraging in the information environment. We believe that our socio-technical approach can help directly address the challenge of "prey in pieces" where the foraging paths were too long and disconnected by different topologies. By explicitly integrating humans in the underlying topology, information foragers can exploit a richer set of relationships.

Codebook [14] confirmed that the small-world phenomenon [25] was readily observed in the socio-technical networks built from the software repositories, e.g., any pair of stakeholders would be connected in six-or-less hops. Our findings suggested that the requirements-centric socio-technical graphs are even smaller with relevant nodes surrounded in four-or-less degrees of separation from the traceability forager's question. Meanwhile, our results revealed several common relationships and their compositions. In light of the recent work on collecting practitioners' natural-language requirements queries (e.g., [26]–[28]), the patterns uncovered by our study could be used to better classify and answer project stakeholders' traceability needs.

Automated requirements traceability tools have been built predominantly by leveraging text retrieval methods [23]. These tools neglect an important factor—familiarity—which plays a crucial role in tracking the life of a requirement. Our work further points out that familiarity is multi-faceted: it could be the person familiar with the issue, the project, or the asker that is knowledgeable about the traceability information. Our results here are to be contrasted with the empirical work carried out by Dekhtyar *et al.* [29] showing that experience (e.g., years worked in software industry) had little impact on human analysts' tracing performance. While a developer's overall background may be broad, we feel that the specific knowledge about the subject software system and the latent relationships established with project stakeholders do play a role in requirements tracing. Automated ways of inferring a developer's knowledge degree (e.g., [30]) would be valuable when incorporated in traceability tools.

Requirements traceability serves as a critical case [31] for our investigation into developers' information foraging in a socio-technical environment. This is because, without the traceability information, many software engineering activities cannot be undertaken, such as verification that a design satisfies the requirements, validation that requirements have been implemented, change impact analysis, system level test coverage analysis, and regression test selection [32]. However, requirements traceability is by no means the only case. In fact, due to information foraging theory's parsimony, its constructs (e.g., patch and scent) have been adapted to support a variety of tasks including debugging, refactoring, and software reuse [33]–[35]. The socio-technical patch created by our spreading activation algorithm, therefore, could help developers answer their needs beyond requirements traceability.

VII. Conclusion and Future Work

By considering common relationships connecting a requirements traceability question to its answer, and encoding these relationships into a spreading activation algorithm, we were able to delineate patches for use in understanding context surrounding requirements traceability questions in a socio-technical environment. In this process, we found that traceability questions were answered by users within four degrees of socio-technical separation; these users were typically Frequent Collaborators of the traceability forager or of the creator/assignee of the issue, or Frequent Contributors to the issue. Encoding these relationships as parameters to a spreading activation algorithm resulted in patches of nodes that a traceability forager could traverse, searching for their answer. While simply creating patches including all nodes within four degrees of socio-technical separation would include all answers, the addition of spreading activation created significantly smaller patches.

This method can further be extended within the requirements traceability realm. With only three node types—issues, comments, and people—we were able to generate these patches. With a higher diversity of information, such as code artifacts and commits, or semantic similarity, more nuanced relationships could be determined, and our process could be improved.

Future work could be conducted on the implementation of the algorithm itself, too. Our decay, frequency reward, and patch cutoff parameters were set through observation and trial and error. More sophisticated statistical analyses, like machine learning algorithms, could help better set these parameters. Our method only suggested the first patch for foraging; in reality, a forager will go through several patches in search of their prey. To accommodate this pattern, this work could be extended to multiple-patch creation.

Most importantly, this method could easily be extended to other socio-technical tasks. In socio-technical tasks, such as comments and responses, we anticipate that a graphical analysis would reveal key relationships that could be considered when constructing patches. Applying spreading activation allows knowledge, proximity, and frequency, among other factors, to be encoded into the patch creation.

REFERENCES

[1] P. Pirolli and S. K. Card, "Information foraging in information access environments," in *Conference on Human Factors in Computing Systems (CHI)*, Denver, CO, USA, May 1995, pp. 51–58.

[2] P. Pirolli, "Computational models of information scent-following in a very large browsable text collection," in *Conference on Human Factors in Computing Systems (CHI)*, Atlanta, GA, USA, March 1997, pp. 3–10.

[3] E. H. Chi, P. Pirolli, K. Chen, and J. E. Pitkow, "Using information scent to model user information needs and actions and the web," in *Conference on Human Factors in Computing Systems (CHI)*, Seattle, WA, USA, March-April 2001, pp. 490–497.

[4] D. Piorkowski, A. Z. Henley, T. Nabi, S. D. Fleming, C. Scaffidi, and M. M. Burnett, "Foraging and navigations, fundamentally: developers' predictions of value and cost," in *International Symposium on Foundations of Software Engineering (FSE)*, Seattle, WA, USA, November 2016, pp. 97–108.

[5] J. Lawrance, C. Bogart, M. M. Burnett, R. K. E. Bellamy, K. Rector, and S. D. Fleming, "How programmers debug, revisited: an information foraging theory perspective," *IEEE Transactions on Software Engineering*, vol. 39, no. 2, pp. 197–215, February 2013.

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

[7] P. Mäder, P. L. Jones, Y. Zhang, and J. Cleland-Huang, "Strategic traceability for safety-critical projects," *IEEE Software*, vol. 30, no. 3, pp. 58–66, May/June 2013.

[8] Q. Forgey and A. E. Weaver, "Key moments from Mark Zuckerberg's senate testimony," https://www.politico.com/story/2018/04/10/zuckerberg-senate-testimony-facebook-key-moments-512334?cid=apn, April 2018, accessed: Apr 2018.

[9] P. Pirolli, *Information Foraging Theory: Adaptive Interaction with Information*. Oxford University Press, 2007.

[10] J. Lawrance, R. K. E. Bellamy, M. M. Burnett, and K. Rector, "Using information scent to model the dynamic foraging behavior of programmers in maintenance tasks," in *Conference on Human Factors in Computing Systems (CHI)*, Florence, Italy, April 2008, pp. 1323–1332.

[11] J. Lawrance, M. M. Burnett, R. K. E. Bellamy, C. Bogart, and C. Swart, "Reactive information foraging for evolving goals," in *Conference on Human Factors in Computing Systems (CHI)*, Atlanta, GA, USA, April 2010, pp. 25–34.

[12] D. Piorkowski, S. D. Fleming, C. Scaffidi, L. John, C. Bogart, B. E. John, M. M. Burnett, and R. K. E. Bellamy, "Modeling programmer navigation: a head-to-head empirical evaluation of predictive models," in *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Pittsburgh, PA, USA, September 2011, pp. 109–116.

[13] A. Begel and R. DeLine, "Codebook: social networking over code," in *International Conference on Software Engineering (ICSE)*, Vancouver, Canada, May 2009, pp. 263–266.

[14] A. Begel, Y. P. Khoo, and T. Zimmermann, "Codebook: discovering and exploiting relationships in software repositories," in *International Conference on Software Engineering (ICSE)*, Cape Town, South Africa, May 2010, pp. 125–134.

[15] T. A. Alspaugh and W. Scacchi, "Ongoing software development without classical requirements," in *International Requirements Engineering Conference (RE)*, Rio de Janeiro, Brazil, July 2013, pp. 165–174.

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

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

[18] E. Knauss, D. Damian, J. Cleland-Huang, and R. Helms, "Patterns of continuous requirements clarification," *Requirements Engineering*, vol. 20, no. 4, pp. 383–403, November 2015.

[19] P. Rempel and P. Mäder, "Preventing defects: the impact of requirements traceability completeness on software quality," *IEEE Transactions on Software Engineering*, vol. 43, no. 8, pp. 777–797, August 2017.

[20] W. Scacchi, "Understanding the requirements for developing open source software systems," *IET Software*, vol. 149, no. 1, pp. 24–39, February 2002.

[21] "Apache software foundation," http://www.apache.org, accessed: Apr 2018.

[22] "JBoss family of lightweight cloud-friendly enterprise-grade products," http: //www.jboss.org, accessed: Apr 2018.

[23] M. Borg, P. Runeson, and A. Ardö, "Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability," *Empirical Software Engineering*, vol. 19, no. 6, pp. 1565–1616, December 2014.

[24] O. Gotel and A. Finkelstein, "Contribution structures," in *International Symposium on Requirements Engineering (RE)*, York, UK, March 1995, pp. 100–107.

[25] D. Chakrabarti and C. Faloutsos, "Graph mining: laws, generators, and algorithms," *ACM Computing Surveys*, vol. 38, no. 1, pp. Article 2, March 2006.

[26] P. Pruski, S. Lohar, W. Goss, A. Rasin, and J. Cleland-Huang, "TiQi: answering unstructured natural language trace queries," *Requirements Engineering*, vol. 20, no. 3, pp. 215–232, September 2015.

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

[28] S. Malviya, M. Vierhauser, J. Cleland-Huang, and S. Ghaisas, "What questions do requirements engineers ask?" in *International Requirements Engineering Conference (RE)*, Lisbon, Portugal, September 2017, pp. 100–109.

[29] A. Dekhtyar, O. Dekhtyar, J. Holden, J. H. Hayes, D. Cuddeback, and W.-K. Kong, "On human analyst performance in assisted requirements tracing: statistical analysis," in *International Requirements Engineering Conference (RE)*, Trento, Italy, August-September 2011, pp. 111–120.

[30] T. Fritz, G. C. Murphy, E. R. Murphy-Hill, J. Ou, and E. Hill, "Degree-of-knowledge: modeling a developer's knowledge of code," *ACM Transactions on Software Engineering and Methodology*, vol. 23, no. 2, p. Article 14, March 2014.

[31] R. K. Yin, *Case Study Research: Design and Methods*. Sage Publications, 2003.

[32] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: the study of methods," *IEEE Transactions on Software Engineering*, vol. 32, no. 1, pp. 4–19, January 2006.

[33] S. D. Fleming, C. Scaffidi, D. Piorkowski, M. M. Burnett, R. K. E. Bellamy, J. Lawrance, and I. Kwan, "An information foraging theory perspective on tools for debugging, refactoring, and reuse tasks," *ACM Transactions on Software Engineering and Methodology*, vol. 22, no. 2, p. Article 14, March 2013.

[34] S. S. Ragavan, S. K. Kuttal, C. Hill, A. Sarma, D. Piorkowski, and M. M. Burnett, "Foraging among an overabundance of similar variants," in *Conference on Human Factors in Computing Systems (CHI)*, San Jose, CA, USA, May 2016, pp. 3509–3521.

[35] S. S. Ragavan, B. Pandya, D. Piorkowski, C. Hill, S. K. Kuttal, A. Sarma, and M. M. Burnett, "PFIS-V: modeling foraging behavior in the presence of variants," in *Conference on Human Factors in Computing Systems (CHI)*, Denver, CO, USA, May 2017, pp. 6232–6244.