Departures from Optimality: Understanding Human Analyst's Information Foraging in Assisted Requirements Tracing

Nan Niu*, Anas Mahmoud*, Zhangji Chen*, and Gary Bradshaw†
* Department of Computer Science and Engineering, Mississippi State University, USA
† Department of Psychology, Mississippi State University, USA
niu@cse.msstate.edu, {amm560, zc140}@msstate.edu, glb2@psychology.msstate.edu

***Abstract*—Studying human analyst's behavior in automated tracing is a new research thrust. Building on a growing body of work in this area, we offer a novel approach to understanding requirements analyst's information seeking and gathering. We model analysts as predators in pursuit of prey — the relevant traceability information, and leverage the optimality models to characterize a rational decision process. The behavior of real analysts with that of the optimal information forager is then compared and contrasted. The results show that the analysts' information diets are much wider than the theory's predictions, and their residing in low-profitability information patches is much longer than the optimal residence time. These uncovered discrepancies not only offer concrete insights into the obstacles faced by analysts, but also lead to principled ways to increase practical tool support for overcoming the obstacles.**

***Index Terms*—Traceability, requirements engineering, study of human analysts, information foraging.**

I. Introduction

Following the life of a requirement and tracking the information in the requirements traceability matrix (RTM) are crucial to many software engineering activities, such as verification and validation, risk assessment, and regression test selection [1]. Research has shown that information retrieval techniques can be effectively applied to help recover the traceability information in an automated fashion [1, 2, 3, 4, 5]. In this way, plausible links can be generated between software artifacts, e.g., between requirements and test cases.

One area that traceability is indispensable is the engineering of mission- or safety-critical software systems. For example, the U.S. Federal Aviation Administration's software certifying standard [6] specifies that at each development stage "developers must be able to demonstrate traceability of designs against requirements". Under these circumstances, a human analyst must vet (e.g., browse and validate) the candidate RTM offered by the tool [7]. Vetting is thus a central activity in *assisted requirements tracing*, in which a human analyst engages with an automated tracing tool to perform the assigned tracing task [8].

Studies of human behavior [7, 8, 9, 10] showed that, in interacting with the tracing tool to prepare their final RTM, the analysts made both errors of omission (threw out correct links) and errors of commission (added incorrect links). A thoroughly conducted experiment by Dekhtyar *et al.* [8] tested 11 vetting variables. The results revealed that only the accuracy of the initial RTM and the analyst effort expended in validating offered links had statistically significant effects on the final RTM, while the other 9 factors (e.g., tool used, tracing experience, effort on searching for omitted links, etc.) did not make a difference [8]. A qualitative study by Kong *et al.* [10] provided additional insights into analysts' behavior. For example, all the analysts were observed to make multiple correct decisions in a row, and such correct-decision bouts were interleaved with streaks of incorrect decisions [10].

Currently, empirically observing analysts' behavior is adopted as the main methodology for researching the human factors in assisted requirements tracing. Observational studies are particularly valuable in answering "*what*" questions by uncovering behavioral patterns. However, little is known about "*why*" analysts behave in a certain way and "*how*" to improve the analysts' tracing performance in a principled manner. Addressing these knowledge gaps is of vital importance because, with a deeper theoretical understanding about the fundamental mechanisms underlying the analysts' behavior, the empirical observations can be related more coherently and the key factors can be tested more completely.

In this work, we explore the theoretical underpinning of analyst's requirements tracing behavior based on Pirolli's *information foraging theory* [11]. The theory uses our animal ancestors' "built-in" food-foraging mechanisms [12] to understand human information seeking and gathering in the vastness of the Web. Lawrance and colleagues [13, 14, 15, 16] have recently pioneered the application of information foraging theory to the debugging domain, and presented encouraging results that matched the theory's predictions with the developers' actual code navigations. Building on their influential work, we aim to investigate human analysts' requirements tracing strategies in light of foraging theory's constructs and principles.

This paper reports an exploratory study that examines two of foraging theory's foundational models [11] in the context of tracing: 1) the *diet model* optimizes the decision related to what kinds of information to consume and what to ignore; and 2) the *patch model* determines the optimal time to spend in an information patch. These optimality models allow us to define the behavioral problems that are posed by the requirements tracing environments, and therefore allow us to

978-1-4673-3076-3/13 © 2013 IEEE

ICSE 2013, San Francisco, CA, USA

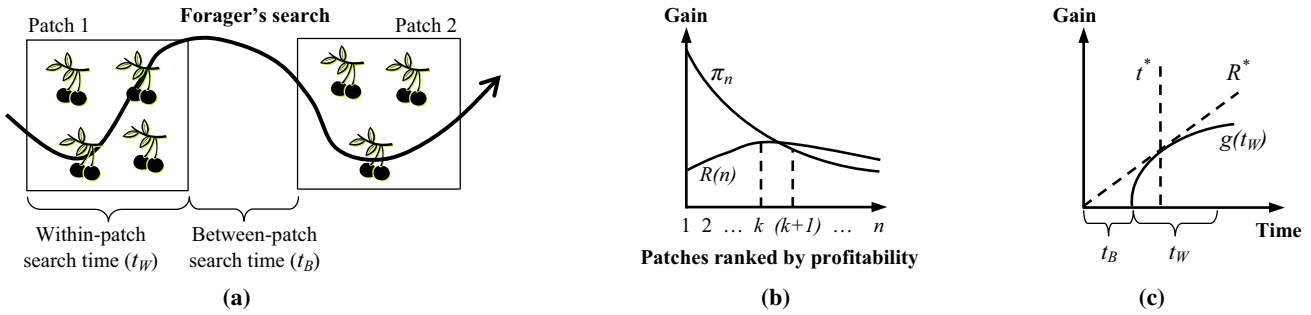


Fig. 1. **(a)** Illustration of patchy environment where a hypothetical bird forages in patches containing berry clusters. **(b)** The optimal diet shall include the k highest profitability prey-patches as the rate of gain, $R(k)$, is greater than the profitability of the next prey-patch, π_{k+1}. **(c)** Charnov's Marginal Value Theorem [12] states that the rate-maximizing time to spend in patch, t^*, occurs when the slope of the within-patch gain function $g(t_W)$ is equal to the average rate of gain, which is the slope of the tangent line R^*.

determine how well an optimal forager (analyst) performs on those problems. We then compare the behavior of real analysts with that of the optimal forager. In particular, the theory's predictions are confronted with the tracing behavior of 6 analysts who validated the links between requirements and code of a software system in the healthcare domain. The departures from optimality revealed by the comparison not only offer concrete insights into the obstacles faced by the human analysts, but also lead to principled ways to overcome the obstacles.

The contributions of our work lie in the analysis of optimality within the shaping limits placed by the task and the information environments. Our work advances the fundamental understanding about analysts' information seeking in light of the adaptiveness of human behavior. This improved understanding, in turn, enables principled ways to increase practical support for software traceability. In what follows, we present background information on foraging theory and assisted requirements tracing in Section II. We then detail our research methodology in Section III. Sections IV and V describe the empirical study's design and results respectively. The implications of our work are discussed in Section VI, and finally, Section VII concludes the paper.

II. Background and Related Work

A. Optimal Foraging Theory

Animals adapt, among other reasons, to increase their rate of energy intake. To do this they evolve different methods: a wolf hunts for prey, but a spider builds a web and allows the prey to come to it. *Optimal foraging theory* is developed in biology for analyzing the adaptive value of food-foraging strategies [12]. Optimality here refers to the strategy that maximizes the gain per unit cost. Central to optimal food foraging are the *diet model* and the *patch model*.

The *diet model* deals with the tradeoffs when a predator forages in an environment in which food is distributed in a patchy manner. Fig. 1a illustrates the patchy environment by presenting a hypothetical bird foraging in berry clusters. The forager must expend some amount of between-patch time (t_B) arriving at a prey-patch, and t_W denotes the within-patch foraging time. In Fig. 1c, $g(t_W)$ represents a decelerating expected net gain function. The amount of energy gained per unit time of foraging is therefore $R = g(t_W)/(t_B + t_W)$. Following Stephens and Krebs [12], we use $\pi = g/t_W$ to denote the patch's profitability and use $\lambda = 1/t_B$ to denote the patch's encounter rate.

The optimal diet selection follows the principle of lost opportunity [12]. Intuitively, the principle states that the prey-patch is predicted to be ignored if its profitability is less than the expected rate of gain of continuing search for other prey-patches. Formally, as shown in Fig. 1b, let the patches be ranked by profitability, $\pi_i = g_i/t_{W\,i}$, such that $\pi_1 > \pi_2 > \ldots > \pi_n$. The optimal diet can be expanded by adding prey-patches in order of decreasing profitability (i.e., from 1 to n) until the rate of gain for a diet of the top k prey-patches is greater than the $k+1$st prey-patch's profitability,

$$\left(R(k) = \frac{\sum_{i=1}^{k} \lambda_i \cdot g_i}{1 + \sum_{i=1}^{k} \lambda_i \cdot t_{W\,i}}\right) > \left(\frac{g_{k+1}}{t_{W\,k+1}} = \pi_{k+1}\right). \quad (1)$$

The left side of the inequality concerns the rate of gain obtained by the diet of the k highest profitability prey-patches, whereas the right side concerns the profitability of the $k+1$st prey-patch. In Fig. 1b, the optimal diet contains {prey-patch$_1$, prey-patch$_2$, …, prey-patch$_k$}, and therefore {prey-patch$_{k+1}$, …, prey-patch$_n$} are predicted to be ignored by the forager.

Once a prey-patch is selected to be part of the forager's diet, the *patch model* deals with predictions of the amount of time to spend within the patch. The basic idea is illustrated in Fig. 1c. As the forager gains energy, the amount of food diminishes or depletes. Consequently, there will be a point at which the expected gains from foraging within the current prey-patch become less than the expected gains that could be made by leaving for a new one. Fig. 1c shows that the rate-maximizing time, t^*, occurs when the derivative of $g(t_W)$ is equal to the slope of the tangent line R^*.

In a nutshell, the simple rule in optimal foraging theory is: "do not expend more energy finding the food than the food provides." Animals (including humans) have evolved some very sophisticated and fascinating food-seeking mechanisms. Optimal foraging theory has been proven to be productive and

resilient in addressing food-searching behavior in the field and the lab, whereby the adequacy of the tenets (e.g., the patch model and the diet model) is tested to account for the evolution of given structures or behavioral traits [12].

B. Foraging Theory Applied to Web and Code Navigation

Humans seeking information adopt various strategies, sometimes with striking parallels to those of animal foragers. The wolf-prey strategy bears some resemblance to classic information retrieval [17], and the spider-web strategy is like information filtering [18]. Pirolli [11] developed *information foraging theory* by laying out the basic analogies between food foraging and information seeking: predator (human in need of information) forages for prey (the useful information) along patches of resources and decides rationally on a diet (what information to consume and what to ignore). By adopting the optimality models and adding details where necessary, Pirolli raised foraging theory from the physical and biological levels to the knowledge and rational levels.

The main application area of information foraging theory is the study of users' information seeking on the Web. During Web navigation, users operate in two environments [11]. The *task environment* embodies a goal, problem, or task that drives human behavior, whereas the *information environment* structures users' interactions with the content. An optimal Web user's navigation is then calculated according to the notion of *information scent* [19], a subjective sense of value and cost of accessing an information source based on perceptual cues. The WUFIS (Web User Flow by Information Scent) algorithm [20] represents one of the most rational and effective computational models of information scent by considering both environments in its computation: 1) a spreading activation network [21] that represents user's goal memory in the task environment; and 2) an inter-word correlation representation used to approximate user's conception of word synonymy [22] in the information environment. Computing the Web user's "information diet" provides remarkable insights into issues like link selection and decision to leave a webpage. As a result, information foraging theory has become extremely useful as a practical tool for website design and evaluation [23].

Inspired by human's adaptive interaction with information on the Web, researchers began to apply foraging theory in software engineering. Ko *et al.* [24] were among the first to relate information foraging to developers' seeking relevant code in software maintenance. In recent years, Lawrance *et al.* [13, 14, 15, 16] have made tremendous strides in understanding programmer navigation during debugging by viewing programmer as predator and bug-fix as prey. Building on Pirolli's work, Lawrance *et al.* [14] developed the PFIS (Programmer Flow by Information Scent) model by combining: 1) a spreading activation over the source code's topology (analogous to links on webpages); and 2) a word similarity measure between the bug report and the source code (computed as vector space model's cosine similarity using the TF-IDF weighting schema [17]). Extending beyond Pirolli's work, Lawrance *et al.* [15] presented the PFIS2 model which incorporated the incremental changes in programmers' conception of the navigation goals during debugging. More recent work [25] focused on empirically assessing programmer navigation models' predictive accuracy and optimally composing single factors (e.g., recency, spatial proximity, etc.) into a family of PFIS3 models.

In summary, information foraging theory [11] provides an evolutionary-ecological approach to understanding human information-seeking on the Web. Applying the theory in software engineering has also been fruitful as the foraging-theoretic approach provides a foundation for studying developers' navigation around the code base [26]. Building and expanding upon Lawrance and colleagues' seminal work on debugging [13, 14, 15, 16], we investigate an important but different information-intensive software engineering task — assisted requirements tracing.

C. Assisted Requirements Tracing

When dealing with a software system's traceability information, a requirements traceability matrix (RTM) establishes the mapping between elements of one artifact (e.g., requirements-level use cases) and elements of the other artifact (e.g., implementation-level classes). Any RTM that exists before the tracing process is complete is called a *candidate* RTM [7]. The *final* RTM is the one approved (or "certified") by a human analyst [7, 10].

Because manually building the RTM can be tedious and error-prone, modern tools employ information retrieval (IR) methods for automated support [1, 2, 3, 4, 5]. These methods search for documents (i.e., candidate traceability links) and retrieve those that are relevant to the query (i.e., element to be traced). Extensive empirical studies have been conducted to evaluate the effectiveness of different IR-based traceability link recovery methods. Converging evidence indicates that all the exploited methods so far are equivalent in that they are able to capture almost the same traceability information [27]. In most cases, a recall of 90% is achievable at precision levels of 5-30% [1, 2, 3, 4, 5, 27, 28], where recall is defined as the percentage of correct links that are retrieved and precision is defined as the percentage of retrieved links that are correct [17].

Although IR-based tools automate the RTM generation to a large extent, in coping with mission- or safety-critical software systems, the human analyst must vet the candidate RTM produced by the tool and add and remove links as necessary to arrive at the final RTM [7]. It is important to emphasize that traceability is not an end in itself but a means towards some other end. The analyst who vets the candidate RTM may be involved in risk assessment, criticality analysis, regulatory compliance, or some other software engineering activities. As a result, the analyst can always override any tool's output and has the final say on whether or not the traceability is correct [9].

Assisted requirements tracing, thus, refers to the process in which the human analyst becomes actively involved and makes

decisions concerning the automated tool's output. The foundational work in this area was laid by Hayes and Dekhtyar [9] where they elucidated the need to study human interaction (reaction) with the tracing tool's results. Since then, a series of studies [7, 8, 10] investigated analyst behavior and revealed that human tended to degrade the accuracy of the RTM provided by the tool. Among the important findings, a rather surprising one was that incorrect decisions were often made if the analyst spent much time on the links [10]. In order to understand the findings like this, it was suggested that we might need to experimentally study one variable at a time [29]. Eleven variables were then examined by Dekhtyar *et al.* [8]. The work represents one of the most significant empirical discoveries to date.

In essence, assisted requirements tracing is aimed at providing the best of both worlds, allowing human and automated tool to do what they do best [8]. While recent empirical contributions have enlightened the vital role of human factors, we believe gaining a theoretical understanding can further advance the field. A foraging-theoretic exploration can shed light on the mechanisms underlying the human's adaptive interaction with the information presented in the tracing tool. This is precisely the focus of our research.

III. Research Methodology

Assisted requirements tracing shares many characteristics with 1) IR-based Web search and 2) navigation along the software entities. As both are domains to which information foraging theory applies (cf. Section II-B), we contend that the analyst's seeking and gathering traceability information can be mathematically modeled in light of the "built-in" foraging mechanisms. In this way, the human analyst can be viewed as a predator in pursuit of prey — the relevant traceability information.

The overall goal of our research is to explore the differences between the analyst's actual information foraging behavior and that defined by the optimality models. Our comparison concentrates on the information diet selection (the diet model) and the residence time within a selected information patch (the patch model). Before formulating specific research questions in Section III-B, we detail how an optimal analyst's decisions are made based on the rational analysis of information foraging.

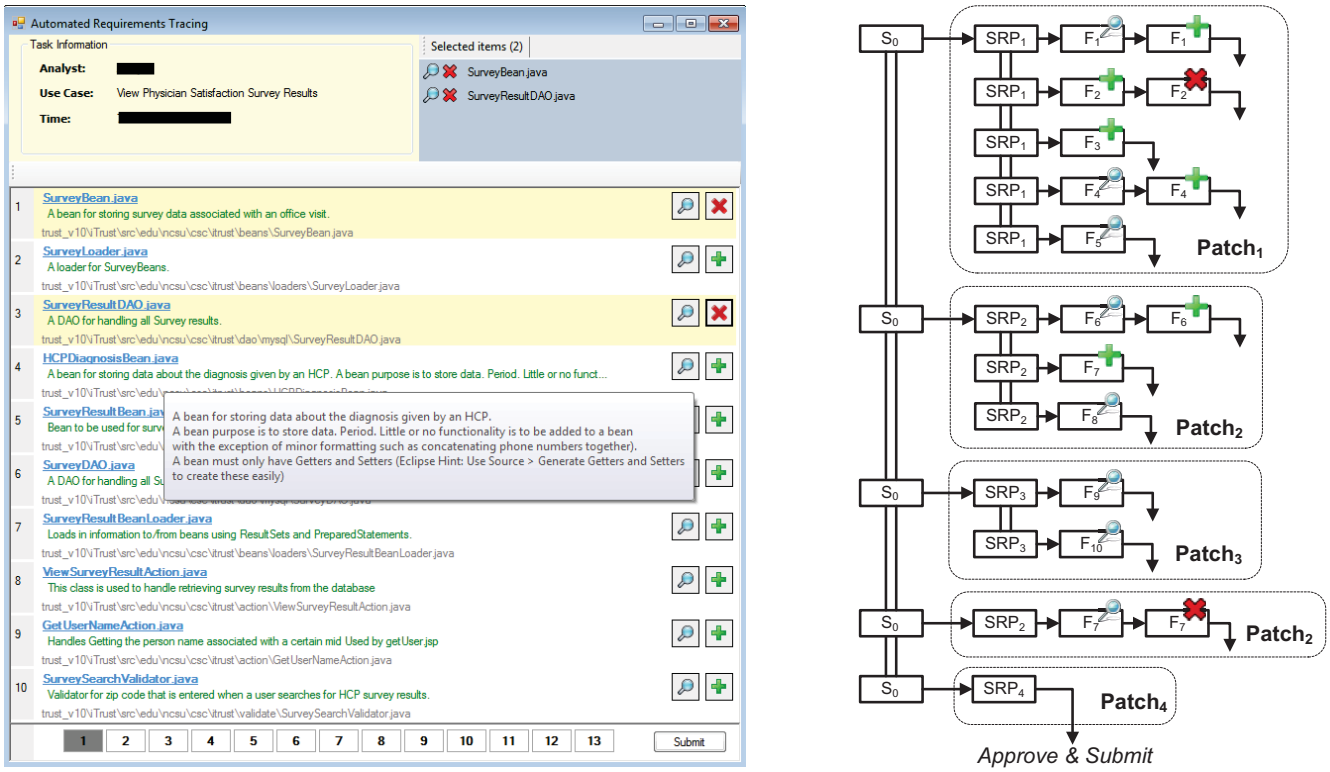
A. Rational Analysis

Anderson's rational analysis [30] is built upon the principle of *optimization under constraints*. The basic idea is that the constraints of the environment place important shaping limits on the optimization that is possible [11]. Applied to debugging, the principle implies that optimal programmers will make the best possible navigational choices, given the information the integrated development environment (IDE) like Eclipse makes available to them at each moment [16]. Similarly, the human analyst's optimal behavior in tracing must be rationalized by scrutinizing the information that the automated tool makes available at each moment.

Fig. 2a shows a screenshot of the automated requirements tracing tool used in our study. We call the tool "ART-Assist" to emphasize the integral yet supportive role it plays in assisted requirements tracing, in which the human analyst must be actively involved [9]. The foremost aim in developing ART-Assist was to obtain functional adequacy of state-of-the-art tracing tools. To that end, we surveyed the basic features of RETRO [1], ADAMS [3], and Poirot [31], and also reviewed our own experience from building the TraCter tool [32]. As different IR-based traceability recovery methods show comparable performance [27], the back-end of ART-Assist adopts the vector space model with TF-IDF weighting [1, 2]. The front-end uses the ordered list to display the retrieved traceability links according to the similarity score computed by the IR algorithm. ART-Assist ranks traceability links much like search engines like Google rank search results in response to a user's query. To support the best practice of *in-place traceability* [33] which advocates tracing-related artifacts being managed within their native environments, each page of ART-Assist presents 10 retrieved links (or more accurately, the links' snippets) as if they reside in Google's search result page (SRP).

Several design decisions about ART-Assist's information handling are worth discussing. We restrict the discussion to the RTM between requirements-level use cases and implementation-level classes, as this is the granularity level at which our empirical study is conducted.

- *What information is used for retrieval?* ART-Assist extends our indexer [34] to process both source code and use case descriptions. Three steps are carried out: tokenizing, filtering (i.e., removing stop words like `the` and `int`), and stemming (i.e., reducing a word to its inflectional root: "patients" → "patient"). The resulting indices, which contain the artifacts' partial and important information, are then used for retrieval.
- *How much retrieved traceability information to keep?* A basic tenet of IR-based methods is that the list of retrieved links contains a higher density of correct traceability links in the upper part of the list and a much lower density of such links in the bottom part of the list [35]. ART-Assist therefore presents the human analyst with 70% of the most similar links, a threshold used in prior work [34].
- *What information is included in the snippet?* The snippet design is informed by our recent study on *topical locality* [36] where we found that class name along with header comments conveyed class body's topic. In addition, the class path offers the file hierarchy information and can act as the URL for the retrieved link. Thus, ART-Assist displays a 3-line snippet in the SRP: the class name, the class header trimmed in 1 line, and the class path. An innovative design in ART-Assist is that a mouse-hover over a link's snippet pops up the full class header comments, as shown in Fig. 2a.
- *How to interact with the traceability information?* ART-Assist's interaction design philosophy is to fulfill ana-



(a) **(b)**

Fig. 2. **(a)** Screenshot of ART-Assist, the automated requirements tracing tool used in our study. To protect anonymity, the analyst's name and the actual tracing time are not released. **(b)** Problem behavior graph that demonstrates the proper mapping of each ART-Assist's page to an information patch. Time in the graph proceeds left to right and top to bottom. Boxes are states. Arrows are moves (transitions). Double vertical lines are returns to a previous state. Dotted enclosing box shows the patch's boundary. The states S_0, SRP_i, and F_j represent the initial state, the i^{th} search result page (SRP), and the information item (traceability link) respectively. Every F_j state is annotated with an ART-Assist icon that indicates the specific operation performed by the human analyst: "magnifying glass" means "view", "+" means "select", and "×" means "deselect". F_j: link (source code file) correspondences are as follows: F_1: SurveyBean.java, F_2: SurveyResultBean.java, F_3: SurveyResultDAO.java, F_4: HCPDiagnosisBean.java, F_5: GetUserNameAction.java, F_6: PersonnelBean.java, F_7: Role.java, F_8: ViewVisitedHCPsAction.java, F_9: PrescriptionReportBean.java, and F_{10}: ViewOfficeVisitAction.java.

lyst's tracing goal while keeping the operations straightforward, accessible, and responsive. Direct navigation to a certain SRP is enabled by clicking the corresponding page number. The highlighted page number shows which SRP is currently displayed. Clicking the class name in the snippet or the "magnifying glass" icon allows the entire class file to be viewed in a new window. A link can be selected or deselected via "+" (add) or "×" (remove). A shopping-cart-like area in ART-Assist's upper-right corner enables the explicit management of selected links. Once a link is selected, its snippet is yellow highlighted. Once the final RTM is approved, the "submit" button shall be pressed.

Understanding how ART-Assist works is necessary for modeling the way the information environment structures an optimal analyst's foraging. Due to the information needs of the human analyst, navigation in ART-Assist has two fundamental differences from both navigation on the Web and navigation in an IDE such as Eclipse. First, what counts as a hyperlink is well-defined in a website [11] and can be readily modeled by the one-click link built in Eclipse [14, 15]. In contrast, only a single hyperlink type is defined in ART-Assist, namely, the click that enables the viewing of the complete source code file. Second, Web and program navigations can be of great depth because many information items can be reached via clickable hyperlinks. In contrast, the depth of ART-Assist navigations is rather limited because the analyst is primarily interested in viewing and (de-)selecting a traceability link. Such differences suggest that the hyperlink topology in a website or a modern IDE is much richer and denser than the one defined in ART-Assist. For this reason, we apply foraging theory's optimality models directly instead of adopting the spreading activation technique used in WUFIS [20] and PFIS [14].

We model each page retrieved by ART-Assist as an *information patch* in which the prey might hide [15]. A patch then contains the SRP and the enclosed *information items* (traceability links) that can be viewed and collected. The key for instantiating "patch", one of foraging theory's core constructs, is to preserve the *locality* such that there are more transitions within a patch than between patches. In Web navigation [11], for instance, a website is treated as an information patch since it is easier to navigate information within the same patch (website) than to navigate information across patches (websites). To assess our patch selection, Fig. 2b uses a

problem behavior graph to visualize an analyst's interaction with ART-Assist when tracing the use case shown in Fig. 2a ('View Physician Satisfaction Survey Results'). A problem behavior graph, which is the foundation for Web behavior graph [11] and code navigation graph [26], is particularly good at showing the *structure* of human's problem solving. Fig. 2b not only illustrates ART-Assist navigation's limited depth, but also supports patch selection's locality property. Among the total of 36 sessions in our empirical study presented later, the average ratio of within- to between-patch operations (transitions) is 4.2. This ratio is comparable to the values found in Web navigation (ratio=3.7 [11]) and in code navigation (ratio=4.4 [26]).

Fig. 3 illustrates how to apply the principle of lost opportunity [11] to determine an optimal forager's information diet. For each of the 5 patches given in Fig. 3a, the information gain (g) is defined by precision. To characterize the within-patch foraging time (t_W), we leverage MAP (mean average precision), a metric widely accepted in the IR community. MAP measures "the quality across the recall levels" [17]. The formal definition of MAP can be found in the standard IR reference [17] and the traceability-specific literature [37]. Intuitively, the higher the MAP, the closer the correct links are to the top of the information patch and therefore the less within-patch foraging time (t_W) an optimal analyst would need. Thus, we define an information patch's profitability as: $\pi = g/t_W = \text{precision} \cdot \text{MAP}$. The optimal diet can then be selected according to the relationship specified in equation (1); here, the encounter rate of all patches is assumed to be a single unit (i.e., λ=1) as ART-Assist allows each patch to be accessed by a single click on the page number. Fig. 3b shows how the theoretical diet (D_Th) is iteratively expanded and optimized in order to counteract the lost opportunity [11].

To analyze the optimal residence time within a patch, we apply Charnov's Theorem [11, 12] to examine how much information value the forager acquires over time t:

$$g(t) = \frac{N_R \cdot t}{N_T \cdot t_s + N_R \cdot t_h}. \quad (2)$$

In this equation, N_R is the number of relevant information items the forager handles, N_T is the total number of information items encountered, t_s is the scanning time, and t_h is the handling time. We instantiate the value of these parameters based on analysts' actual tracing sessions. This allows for theoretically determining the optimal within-patch residence time (cf. Fig. 1c) which we denote by t*_Th.

B. Research Questions

The research questions addressed in our work are the differences and discrepancies between D_Th (optimal diet in theory) and D_Ac (analyst's actual information diet), and those between t*_Th (optimal within-patch residence time in theory) and t*_Ac (analysts' actual within-patch residence time). Specifically, we are interested in understanding real

Patch	**Precision** (g)	**MAP** ($1/t_W$)	**Profitability** (π)
Patch_1	0.40	0.67	0.27
Patch_2	0.30	0.81	0.24
Patch_3	0.20	1.00	0.20
Patch_4	0.20	0.83	0.17
Patch_5	0.10	1.00	0.10

(a)

k	$R(k)$	π_{k+1}	D_Th (optimal diet in theory)
0	–	–	$\{\text{Patch}_1\}$
1	0.16	0.24	$\{\text{Patch}_1, \text{Patch}_2\}$
2	0.19	0.20	$\{\text{Patch}_1, \text{Patch}_2, \text{Patch}_3\}$
3	0.19	0.17	$\{\text{Patch}_1, \text{Patch}_2, \text{Patch}_3\}$

// Following equation (1), the optimal diet selection
// terminates because $R(3) > \pi_4$.

(b)

Fig. 3. **(a)** Optimal diet selection considers information patches in order of decreasing profitability. **(b)** The optimal diet (D_Th) is obtained by iteratively adding Patch_k as long as the gain of the diet is not greater than the profitability of the next patch, $R(k) \leq \pi_{k+1}$.

analysts' deviations and departures from optimality from two complementary perspectives.

- *Structural*. To what extent is an information patch's selection affected by its profitability and quality? While profitability ($\pi = \text{precision} \cdot \text{MAP}$) is computed only if the answer set of true links is known, quality of a patch can be measured by its internal cohesion [38].
- *Behavioral*. To what degree is an information patch's selection affected by the residence time and the navigation behavior of the human analysts?

The answers to the research questions will enable not only a systematic assessment of the factors suggested by information foraging theory, but also a coherent account for unifying the observations that would otherwise not be linked in a meaningful way. For example, with the foraging-theoretic foundation we speculate that (i) the interleave of analyst's correct- and incorrect-decision streaks [10] might be due to the inclusion of both optimal and non-optimal information patches in the actual diet, and (ii) the incorrect decisions after a long foraging period [10] might be attributed to the excessive residence time within a patch.

IV. Empirical Study Setup

To answer the research questions, we conducted an assisted requirements tracing experiment by using the iTrust dataset (http://agile.csc.ncsu.edu/iTrust). iTrust is a Java application aimed at providing patients with a means to keep up with their medical records, as well as to communicate with their doctors. Although originated as a course project, iTrust has exhibited real-world relevance and served as a traceability testbed for understanding the importance of security and privacy requirements in the healthcare domain [39].

The iTrust dataset has 46 use cases (UCs) and 226 Java classes. The requirements-to-source-code traceability matrix is of size 10,396. The answer set, prepared by iTrust's developers, contains 314 true links. Since we wanted to observe

TABLE I
REQUIREMENTS TRACES USED IN THE EXPERIMENT

ID (our study)	Title (iTrust ID)	|true links|	|true links retrieved|
UC_1	Document Office Visit (UC-11)	26	25
UC_2	Maintain a Hospital Listing (UC-18)	4	4
UC_3	Maintain Standards Lists (UC-15)	13	12
UC_4	Safe Drug Prescription (UC-37)	20	17
UC_5	View Physician Satisfaction Survey Results (UC-25)	8	8
UC_6	View Patients (UC-28)	4	4

analyst's navigation behavior, only the UCs with more than 1 true link defined in the answer set were considered. We identified 32 such UCs, among which 6 were randomly selected. Table I lists these requirements tracing tasks. Note that ART-Assist keeps 70% of the most similar links in the retrieval results. Table I shows that, in some cases, not every true link is presented in ART-Assist.

We recruited 6 upper-division students in computing science from Mississippi State University, including 2 seniors and 4 graduate students. None of the participants knew iTrust before the experiment, but all of them reported being familiar with the healthcare domain. The participants had all learned about traceability and reported a median of 0.25 years tracing experience (mainly done manually).

During the experiment, each participant (analyst) worked alone in a lab and began by signing the consent form and by learning how to use the ART-Assist tool. The analyst was then given hard copies of the UC descriptions and was told to use only ART-Assist and not to use internet or any other resources in the experiment. We asked the analyst to trace all 6 UCs and to carry out the tracing tasks in any order they would prefer. A researcher was present to run the ART-Assist tutorial, to encourage the analyst to think aloud during tracing, and to conduct an informal exit interview to elicit the analyst's feedback about their tracing experience. Each experiment session lasted approximately 1 hour.

V. RESULTS AND ANALYSIS

ART-Assist logs fine-grained, time-stamped user interactions. In order to extract the analyst's actual diet (D_Ac) from the logs, we analyze the task environment by modeling the problem space of tracing. Fig. 4 shows the state-transition diagram that depicts the lifecycle of an information item (namely, a traceability link). The analyst may view (scan) the item as it is presented in ART-Assist's retrieval page (patch), and further pursue the prey (link) if she regards it as relevant. The analyst may refine an item's (de-)selection for several times, during which the item can be viewed optionally. Upon analyst's approval, the item becomes part of the finally submitted RTM.

Modeling the task environment is critical to defining the variables of rational analysis (cf. Section III-A), especially those appeared in equation (2). In our study, t_s (scanning time) denotes the view time for an item that is not selected. In another word, t_s refers to the time the leftmost, self-looped "*View*" transition in Fig. 4 takes. The other 4 transitions in

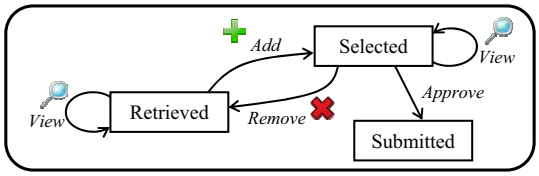


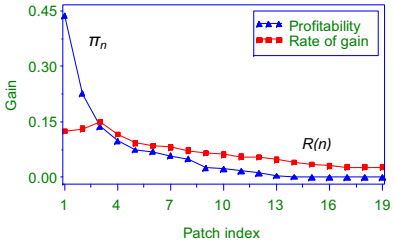
Fig. 4. State-space of an information item (traceability link) as a human analyst with the tracing task interacts with the ART-Assist tool. Boxes show states. Arrows show state transitions annotated with icons representing the ART-Assist operations (cf. Fig. 2).

Fig. 4 represent operations on what the analyst believes to be a relevant link. Thus, the time transitioned to and from the "Selected" state in Fig. 4 gives rise to t_h — the handling time. Based on this, we define the analyst's actual diet to be the set of patches containing handled items, i.e., D_Ac = {Patch $P \mid \exists$ link $l \in P$ such that 'the analyst handles l as a relevant link at some point during tracing'}.

For the comparison of D_Ac with D_Th, Fig. 5a shows the average case in which the trace's information environment shapes the optimal diet selection. Fig. 5b provides D_Th specific to each UC, along with the total number of patches (pages) retrieved by ART-Assist. On average, only 13.5% of the available patches are included in D_Th. In contrast, D_Ac is much less selective as it is composed of an average of 45.0% available patches. For all 6 tracing tasks, D_Th is a proper subset of D_Ac, which implies that the theory's predictions are highly accurate. To evaluate the matching degree with D_Th, we expand upon the work of Lawrance *et al.* [13] and use the analysts' consensus to further categorize the patches in D_Ac.

- *Match* refers to the overlap between D_Ac and D_Th. We define the match is *large* if the patch appears in over half of the analysts' actual diets (i.e., $\geq$3 analysts' diets in our study); otherwise, the match is *slight*.
- *Departure* refers to the difference of D_Ac and D_Th. We say the departure is *large* if the patch is handled by over half of the analysts as they pursue relevant prey in the patch; otherwise, the departure is *slight*.

Table II shows the structural and behavioral aspects of requirements analyst's information foraging. Descriptive statistics are given in terms of $(\text{mean} \pm \text{standard deviation})$. Inferential statistics are performed via the Mann-Whitney test [40], a non-parametric test which was also used by Lawrance *et al.* [13] for assessing software developer's information foraging. It is evident from Table II that the matched diets are more profitable than the departed ones. This should not be surprising since the optimal diet (D_Th) is selected based on the descending order of profitability (π). However, the analysts did pursue in a greater number of low-profitable prey-patches than in theoretically high-profitable ones (Mann-Whitney, U=32.5, $p<0.05$). On one hand, this may account for the quality degradation on the RTM after human vetted the retrieved links [7, 8, 9, 10]. On the other hand, this may suggest that analysts needed to consume the "bad" in order to recognize the "good". In this sense, the correct and incorrect decisions were indeed interdependent.



(a)

Use Case	# of patches (total)	D_Th (optimal diet)	D_Ac (human analysts' actual information diets)			
			Largely matched with D_Th	Slightly matched with D_Th	Slightly departed from D_Th	Largely departed from D_Th
UC_1	17	$\{P_1, P_2, P_3\}$	$\{P_1\}$	$\{P_2, P_3\}$	$\{P_6, P_8, P_{11}, P_{12}, P_{16}\}$	$\{P_4, P_5\}$
UC_2	19	$\{P_1\}$	$\{P_1\}$	Φ	$\{P_3, P_4, P_{11}, P_{19}\}$	$\{P_2\}$
UC_3	15	$\{P_1, P_2, P_3\}$	$\{P_1, P_2, P_3\}$	Φ	$\{P_7, P_8, P_{13}\}$	$\{P_4, P_6\}$
UC_4	12	$\{P_1, P_2\}$	$\{P_1, P_2\}$	Φ	$\{P_4\}$	$\{P_3\}$
UC_5	13	$\{P_1\}$	$\{P_1\}$	Φ	$\{P_2, P_3, P_4, P_5\}$	Φ
UC_6	14	$\{P_1, P_2\}$	$\{P_1\}$	$\{P_2\}$	$\{P_5, P_6, P_7, P_{13}\}$	$\{P_3, P_4\}$

(b)

Fig. 5. **(a)** Applying the diet model (cf. Fig. 1b) to the experimental tracing tasks. $R(n)$ and π_n are both averaged over the 6 UCs. **(b)** Comparing optimal forager's diet (D_Th) with real analysts' diets (D_Ac). P_i denotes $Patch_i$ representing the i^{th} page retrieved by ART-Assist.

TABLE II
ASSESSING HUMAN ANALYST'S INFORMATION FORAGING FROM STRUCTURAL AND BEHAVIORAL PERSPECTIVES

D_Ac Categories		Structural Properties		Navigational Behaviors (all time values are in seconds)				
		π (profitability)	cohesion	# of revisits	t_s	t_h	t*_Ac	Δt*= t*_Ac−t*_Th
Match with D_Th	Large	0.32 ± 0.13	0.61 ± 0.25	0.45 ± 0.09	6.11 ± 1.30	4.95 ± 1.08	29.50 ± 22.81	9.55 ± 3.01
	Slight	0.27 ± 0.18	0.40 ± 0.34	0.33 ± 0.07	6.05 ± 1.13	6.27 ± 2.85	35.29 ± 17.34	13.19 ± 5.53
Departure from D_Th	Slight	0.09 ± 0.06	0.35 ± 0.19	1.94 ± 1.38	7.97 ± 2.18	9.58 ± 3.44	90.03 ± 46.72	56.07 ± 24.49
	Large	0.12 ± 0.07	0.53 ± 0.21	1.05 ± 0.63	7.31 ± 2.69	8.08 ± 3.14	55.64 ± 23.26	48.40 ± 31.72

Profitability (π) is computed based on the answer set that defines the true links for each requirement. Under non-experimental settings where no answer set is available, Duan and Cleland-Huang [38] argued that internal metrics, such as coupling and cohesion, could be used to assess the quality of the cluster-patch. We adapt this idea and compute the *patch cohesion* as the average pairwise TF-IDF differences of all the information items in a given patch. Table II shows that greater consensus (largely matched and largely departed) was achieved on more cohesive patches. Further comparison reveals that the cohesion of analysts' handled patches (i.e., those in D_Ac) is significantly greater than that of the patches the analysts did not view as relevant (Mann-Whitney, U=271.0, p<0.01). Thus, analysts seem to use cohesion to judge an information patch's relevance; testing this hypothesis requires future research.

As far as the navigation behavior is concerned, the patches matched with D_Th were visited mostly once. However, the D_Th-departed patches received a surprisingly high number of revisits. This shows analysts' struggles with deciding the relevance of certain prey-patches. Even when relevance was determined, the struggles with D_Th-departed patches continued as analysts expended more time handling the links (t_h). Interestingly, the scanning time (t_s) stayed roughly the same across the D_Ac categories.

Our final analysis is concerned with analysts' within-patch residence time (t*_Ac) and how it differs from the optimal residence time (t*_Th). As shown in Table II, t*_Ac exhibits considerable variation among D_Ac categories, but in all the cases, the actual residence time is greater than t*_Th. Such deviations from optimality (Δt*) were observed to be substantially greater in D_Th-departed diets than in D_Th-matched ones (Mann-Whitney, U=744.5, p<0.01). Fig. 6 uses the navigation steps to illustrate Δt*. To reduce clutter, only one sample Δt* (largely matched) is given in Fig. 6.

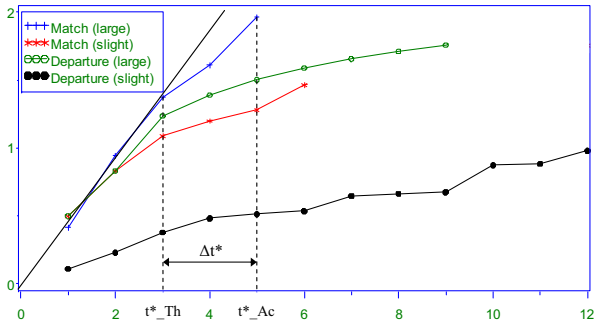


Fig. 6. Applying the patch model (cf. Fig. 1c) to plot the average information gain (y-axis) per navigation step (x-axis). The value of the $<t_s, t_h>$ pair (cf. equation (2)) is instantiated by the average time according to the analysts' actual navigations reported in Table II.

VI. DISCUSSION

As our results indicate that discrepancies exist between human analyst's information seeking and the behavior determined via foraging theory's optimality models, we suggest design guidelines for tools supporting analysts in performing requirements tracing. We then relate our work to other models and discuss the limitations of our study.

A. Implications for Tool Support

Our objective of uncovering the gaps from optimality is to enable principled ways to reduce the gaps. Our study shows several important discrepancies that provide concrete insights into the behavioral traits of and the obstacles faced by human analysts.

First, although the low-profitable, non-optimal patches (i.e., D_Th-departed patches) turned out to be indispensable for analysts' information diets, the analysts did struggle to determine the relevance of those patches. The primary reason, based on our interviews, was the lack of contextual information when the analysts navigated from one patch (page) to another. In fact, the analysts often unintentionally returned to locations

that had already been visited. Most agreed that such revisits were wasted interactions since they had to repeat the relevance judgments. One way to alleviate the struggles is to introduce explicit tagging or rejecting the patch as a whole, as designed in the work of Duan and Cleland-Huang [38]. Another support is to leverage advanced information scent modeling techniques, such as exploiting analyst's navigation recency [15], to generate reactive navigation recommendations.

Second, while the scanning time (t_s) remained approximately constant, the handling time (t_h) was longer for D_Th-departed diets than for D_Th-matched ones. During tracing, more than half of the analysts expressed uncertainty about whether a link should be placed in the shopping-cart for checkout. Some suggested that the gathering of traceability information could be diversified. For example, a black list of irrelevant links and a working area for storing to-be-determined links could be added to the tracing tool.

Third, the analysts invariably overspent the within-patch time (t^*_Ac $> t^*$_Th), especially when foraging in D_Th-departed patches. Enabling analysts to correctly reason about the patch profitability can help them to shorten the time difference (Δt^*). However, it is not possible to expect the analysts to have the perfect knowledge about the information environment. Our study implies that patch cohesion, one of the internal quality indicators, can be of much practical value. In this way, cohesion acts as the *perceived* profitability [13] and its improvement via clustering [38, 41] has already led to remarkable enhancements in tracing.

B. Relationship to Other Models

Models of Information Seeking and Gathering. Ko *et al.* [24] suggested that software maintainers' seeking relevant code follows an iterative 'Search-Relate-Collect' process. While in line with Ko's model, assisted requirements tracing is also related to Web search (e.g., the initiation, selection, and collection stages described by Hearst [42]). Although it is well known that Web users view only the first few search result pages (e.g., Jansen *et al.* [43] reported that 80% of the users viewed only the first 2 pages), our results show that human analysts did go as far as the last page to collect the relevant traceability information.

Foraging-Theoretic Approaches to Code Navigation. Table III situates our study within the previous research investigating programmer navigation. Because tracing is a new domain for applying foraging theory, our mapping of an information patch is different from the prior work. The most important difference, in our opinion, is the use of the diet model in our study to determine D_Th, which represents a novel mechanism for assessing D_Ac.

Studies of Human Factors in Tracing. Our work complements the studies of analyst's tracing performance based on the final RTM's quality (e.g., [7, 8, 9]). In our study, both the RTM decision and the rational decision-making process are examined. While our work demonstrates the value of optimal foraging models, how to expand the analysis to account for the predictable level of human fallibility [29] and to balance the economics of maintaining and utilizing the requirements traces [44, 45] remains an open question.

TABLE III
COMPARING OUR WORK WITH OTHER FORAGING-THEORETIC MODELS

Fundamentals of information foraging theory	**Patches**	**Scent**	**Diet**
Seminal work [13]	Classes	Textual similarity	D_Ac
PFIS [14]	Classes	Textual similarity, Program topology	D_Ac
PFIS2 [15]	Methods	Program topology, Navigation recency	D_Ac
PFIS3 [25]	Methods	Single factors, Optimal composites	D_Ac
Our work	Pages retrieved	Textual similarity	D_Ac, D_Th

C. Study Limitations

The applicability of this study's results may be limited by ART-Assist's design, operationalization of analyst's actual diet, and participants' unfamiliarity with the subject system.

ART-Assist provides basic features commonly found in IR-based tracing tools. The 70% threshold filters out certain true links. Adjusting this value, statically or dynamically, may alter the analyst's information foraging behavior. A similar limitation applies to structuring 10 links per page.

When defining D_Ac, we adopted a behavioral viewpoint by focusing on *how* to operate a link (cf. Fig. 4) rather than *what* links were approved in the end. Shifting D_Ac's definition to the final RTM's perspective would modify an important assumption of the decision problem. Once assumptions like this are updated, they can feed back into the rational analysis of the information forager.

Our work with student participants limits how the results could be generalized. Egyed *et al.* [45] note that in many industrial settings people have no intimate system knowledge during trace recovery. There is also precedence in traceability work: prior studies have used students with low levels of industry experience to represent new people joining a company [7, 8, 10, 45]. Nevertheless, it would be interesting to study how familiarity levels may alter the tracing behavior.

VII. CONCLUSIONS

The main contributions of this paper are the evolutionary-ecological understanding of the fundamental mechanisms underlying human analysts' requirements tracing behaviors, the theoretical analysis of optimality within the shaping limits placed by tracing's task and information environments, the empirical evaluation of the matches and mismatches between theory's predictions and analysts' actual behaviors, and the concrete insights of the principled ways to increase practical support for software traceability.

Building on the extensive research on the IR-based candidate link recovery methods [1, 2, 3, 4, 5], the study of human analysts represents a milestone in the traceability literature, as we now have reached a general consensus regarding the equivalence of the underlying IR methods [27]. The success of requirements tracing, as measured by the final RTM's quality,

therefore hinges largely on the analysts' interactions with and decisions about the tool's output. Building on the growing body of work on human factors [7, 8, 9, 10], it is hoped that our work contributes a step towards understanding the ecologically valid ways to "design a fast, accurate and certifiable tracing process" [29].

ACKNOWLEDGEMENT

We thank all the participants of our study, as well as Tanmay Bhowmik and Sandeep Reddivari for comments on earlier drafts of this paper. The work is funded by the U.S. NSF (National Science Foundation) Grant CCF-1238336 and the Mississippi State University's Cross-disciplinary Research Facilitation Grant Program.

REFERENCES

[1] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: the study of methods," *IEEE TSE*, vol. 32(1), pp. 4–19, 2006.

[2] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE TSE*, vol. 28(10), pp. 970–983, 2002.

[3] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM TOSEM*, vol. 16(4), 2007.

[4] X. Zou, R. Settimi, and J. Cleland-Huang, "Improving automated requirements trace retrieval: a study of term-based enhancement methods," *Empir Softw Eng*, vol. 15(2), pp. 119–146, 2010.

[5] X. Chen and J. Grundy, "Improving automated documentation to code traceability by combining retrieval techniques," in *ASE*, 2011, pp. 223–232.

[6] U.S. Federal Aviation Administration, "RTCA/DO-178B Software Considerations in Airborne Systems and Equipment Certification," 1992.

[7] D. Cuddeback, A. Dekhtyar, and J. H. Hayes, "Automated requirements traceability: the study of human analysts," in *RE*, 2010, pp. 231–240.

[8] 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 *RE*, 2011, pp. 111–120.

[9] J. H. Hayes and A. Dekhtyar, "Humans in the traceability loop: can't live with 'em, can't live without 'em," in *TEFSE*, 2005, pp. 20–23.

[10] W.-K. Kong, J. H. Hayes, A. Dekhtyar, and J. Holden, "How do we trace requirements? an initial study of analyst behavior in trace validation tasks," in *CHASE*, 2011, pp. 32–39.

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

[12] D. W. Stephens and J. R. Krebs, *Foraging Theory*. Princeton University Press, 1986.

[13] J. Lawrance, R. Bellamy, and M. Burnett, "Scents in programs: does information foraging theory apply to program maintenance?" in *VL/HCC*, 2007, pp. 15–22.

[14] J. Lawrance, R. Bellamy, M. Burnett, and K. Rector, "Using information scent to model the dynamic foraging behavior of programmers in maintenance tasks," in *CHI*, 2008, pp. 1323–1332.

[15] J. Lawrance, M. Burnett, R. Bellamy, C. Bogart, and C. Swart, "Reactive information foraging for evolving goals," in *CHI*, 2010, pp. 25–34.

[16] J. Lawrance, C. Bogart, M. Burnett, R. Bellamy, K. Rector, and S. D. Fleming, "How programmers debug, revisited: an information foraging theory perspective," *IEEE TSE*, (accepted).

[17] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[18] U. Shardanand and P. Maes, "Social information filtering: algorithms for automating "word of mouth"," in *CHI*, 1995, pp. 210–217.

[19] P. Pirolli, "Computational models of information scent-following in a very large browsable text collection," in *CHI*, 1997, pp. 3–10.

[20] E. H. Chi, P. Pirolli, K. Chen, and J. E. Pitkow, "Using information scent to model user information needs and actions on the Web," in *CHI*, 2001, pp. 490–497.

[21] J. R. Anderson and P. Pirolli, "Spread of activation," *Journal of Experimental Psychology*, vol. 10, pp. 791–798, 1984.

[22] H. Schütze, "Dimensions of meaning," in *SC*, 1992, pp. 787–796.

[23] J. M. Spool, C. Perfetti, and D. Brittan, *Designing for the Scent of Information*. User Interface Engineering, 2004.

[24] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," *IEEE TSE*, vol. 32(12), pp. 971–987, 2006.

[25] 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 *VL/HCC*, 2011, pp. 109–116.

[26] N. Niu, A. Mahmoud, and G. Bradshaw, "Information foraging as a foundation for code navigation," in *ICSE*, 2011, pp. 816–819.

[27] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, "On the equivalence of information retrieval methods for automated traceability link recovery," in *ICPC*, 2010, pp. 68–71.

[28] J. H. Hayes and A. Dekhtyar, "A framework for comparing requirements tracing experiments," *IJSEKE*, vol. 15(5), pp. 751–782, 2005.

[29] D. Cuddeback, A. Dekhtyar, J. H. Hayes, J. Holden, and W.-K. Kong, "Towards overcoming human analyst fallibility in the requirements tracing process (NIER Track)," in *ICSE*, 2011, pp. 860–863.

[30] J. R. Anderson, *The Adaptive Character of Thought*. Lawrence Erlbaum Associates, 1990.

[31] J. Lin, C. C. Lin, J. Cleland-Huang, R. Settimi, J. Amaya, G. Bedford, B. Berenbach, O. B. Khadra, C. Duan, and X. Zou, "Poirot: a distributed tool supporting enterprise-wide automated traceability," in *RE*, 2006, pp. 356–357.

[32] A. Mahmoud and N. Niu, "TraCter: a tool for candidate traceability link clustering," in *RE*, 2011, pp. 335–336.

[33] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova, "Best practices for automated traceability," *IEEE Computer*, vol. 40(6), pp. 27–35, 2007.

[34] A. Mahmoud and N. Niu, "Source code indexing for automated tracing," in *TEFSE*, 2011, pp. 3–9.

[35] A. De Lucia, R. Oliveto, and G. Tortora, "IR-based traceability recovery processes: an empirical comparison of "one-shot" and incremental processes," in *ASE*, 2008, pp. 39–48.

[36] N. Niu, J. Savolainen, T. Bhowmik, A. Mahmoud, and S. Reddivari, "A framework for examining topical locality in object-oriented software," in *COMPSAC*, 2012, pp. 219–224.

[37] H. Sultanov, J. H. Hayes, and W.-K. Kong, "Application of swarm techniques to requirements tracing," *REJ*, vol. 16, pp. 209–226, 2011.

[38] C. Duan and J. Cleland-Huang, "Clustering support for automated tracing," in *ASE*, 2007, pp. 244–253.

[39] A. Meneely, B. Smith, and L. Williams, "iTrust electronic health care system: a case study," in *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds. Springer, 2012.

[40] W. J. Conover, *Practical Nonparametric Statistics*. Wiley, 1999.

[41] N. Niu and A. Mahmoud, "Enhancing candidate link generation for requirements tracing: the cluster hypothesis revisited," in *RE*, 2012, pp. 81–90.

[42] M. A. Hearst, *Search User Interfaces*. Cambridge University Press, 2009.

[43] B. J. Jansen, A. Spink, J. Bateman, and T. Saracevic, "Real life information retrieval: a study of user queries on the Web," *ACM SIGIR Forum*, vol. 32(1), pp. 5–17, 1998.

[44] A. Egyed, P. Grünbacher, M. Heindl, and S. Biffl, "Value-based requirements traceability: lessons learned," in *RE*, 2007, pp. 115–118.

[45] A. Egyed, F. Graf, and P. Grünbacher, "Effort and quality of recovering requirements-to-code traces: two exploratory experiments," in *RE*, 2010, pp. 221–230.