Tagging in Assisted Tracing

Wentao Wang*, Nan Niu*, Hui Liu†, and Yuting Wu†
* Department of Electrical Engineering and Computing Systems, University of Cincinnati, USA
† School of Computer Science and Technology, Beijing Institute of Technology, China
wang2wt@mail.uc.edu, nan.niu@uc.edu, {liuhui08, wuyuting}@bit.edu.cn

***Abstract*—Assisted tracing is the process where human analyst vets and makes decisions concerning the automated method's output. Current research reveals human fallibility in this process, and shows that analyst often makes incorrect decisions that lead to inaccurate final trace matrix. To help enhance analyst performance, we leverage tagging in assisted tracing. Specifically, we implement tagging as a front-end feature that allows analysts to freely mark what they feel worth externalizing during tracing. We then carry out an experiment to investigate the tagging practices of 28 student analysts in vetting requirements-to-source-code trace matrices. Our study shows that tagging is readily adopted by analysts, tags produced in tracing follow power laws, and tags greatly enhance the precision of analyst-submitted final trace matrices. Our work opens up new avenues for researching improved ways to foster analyst-tool integration.**

***Index Terms*—Traceability, assisted tracing, human factors, tagging, front-end of tracing tool, analyst-tool interaction.**

I. Introduction

When dealing with a software system's traceability information, a trace matrix (TM) establishes the mapping between artifacts of one type (e.g., requirements-level use cases) and artifacts of another type (e.g., implementation-level methods). In *assisted tracing*, a human analyst uses an automated method to generate a candidate TM, reviews it, makes any desired changes, and certifies the final TM [1].

State-of-the-art tracing tools, such as RETRO [2], Poirot [3], and Traceclipse [4], employ information retrieval (IR) algorithms at their back ends to generate the candidate TM. While these automated tracing methods strive to retrieve all and only the correct traceability links, the retrieval results still miss correct links and contain incorrect ones. A strength of assisted tracing, then, comes from the engagement of human analyst in fixing the errors made by automated methods.

However, recent studies show that, in interacting with automatically generated candidate TMs, analysts often make incorrect decisions that lead to final TMs of even lower quality [5, 6, 7]. Understanding human fallibility enables us to rethink the many facets of assisted tracing [1, 8, 9, 10]. At the back end, for example, the performance of automated methods cannot be measured only by the candidate TM's accuracy. In addition, qualities like browsability and discernability should be considered in assessing and selecting tracing algorithms [2].

A new direction in assisted tracing research is constructing the tracing tool's front end in such a way that enhances analyst performance [9]. To move toward this direction, we investigate how individual analysts react to *tagging* when built into a tracing tool's front end. Using the tagging feature, analysts can freely mark user-defined keywords during the tracing session. We report in this paper an empirical study on the tagging practices of 28 student analysts engaged in examining and finalizing requirements-to-source-code TMs.

Tagging is a lightweight social computing mechanism that has seen rapid and wide adoption by communities on the Web, e.g., Flickr and YouTube. Practitioners also find tagging useful in software development because it complements the formal engineering practices [11]. It is this *informal* nature of tagging that we want to incorporate into a tracing tool's front end. By providing analysts with an interface to record openly what they feel worth externalizing, we are interested in exploring what tags are produced and how they are used in tracing.

Prior work by Hale and colleagues [12] analyzed the role of tags as traceability links in software systems developed by student teams. Compared to our work, several key differences exist. In [12], tagging was done by the original developers throughout the software lifecycle via a courseware tool. By contrast, analysts in our study are not the original developers and they perform tagging in a tracing-specific tool. The most important distinction, in our opinion, is that Hale *et al.* [12] treated tags themselves as TMs, whereas we regard tags only as a means of supporting analysts for finalizing their TMs.

The contributions of our work lie in the new front end tagging design introduced to the tracing tool and the systematic examination of analyst behavior in response to such a design. Our effort thus represents a crucial first step toward creating more productive analyst-tool interactions in assisted tracing [9]. In what follows, we review related work in Section II. Our research questions are presented in Section III and Section IV discusses our methodology. Section V answers the research questions by detailing the empirical study results, and finally, Section VI concludes the paper.

II. Background and Related Work

A. Assisted Tracing

The study of tracing methods has resulted in much progress toward the automated generation of TMs. Many tracing tools employ IR-based methods to recover plausible links. Evaluated under the basic IR metrics of recall and precision [13], automated tracing methods can generate TMs that include most of the true links that should be found (80-90% recall). These methods, however, also retrieve many false links (below 10% precision on large datasets and 20-40% precision on smaller datasets with unfiltered results) [2, 4, 14, 15, 16].

978-0-7695-5593-5/15 $31.00 © 2015 IEEE
DOI 10.1109/SST.2015.9

Although IR-based tools automate the TM generation to a large extent, in coping with mission- or safety-critical software systems, the human analyst must vet the candidate TM produced by the tool and add and remove links as necessary to arrive at the final TM [5]. It is important to emphasize that traceability is not an end in itself but a means toward some other end. The analyst who vets the candidate TM 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 automated method's output and has the final say on whether or not the traceability is correct [17].

Assisted tracing, thus, refers to the process in which the human analyst becomes actively involved and makes decisions concerning the automated method's output. The foundational work in this area was laid by Hayes and Dekhtyar [17] where they elucidated the need to study human interaction (reaction) with the tracing tool's results. Since then, a series of studies [5, 6, 8] investigated analyst behavior and revealed human fallibility in assisted tracing. More recent work statistically tested the variables that might affect analyst performance [7], analytically defined the measures to capture analyst work progress [1], and theoretically understood the mechanisms underlying analyst behavior [10].

In essence, assisted tracing is aimed at providing the best of both worlds, allowing human analyst and automated method to do what they do best [7]. While current IR-based methods cannot achieve a high recall without also retrieving a great number of false links, a tracing tool that induces better analyst interaction with the candidate TM is of vital importance. This is because, in assisted tracing, the accuracy of the analyst-submitted final TM trumps the accuracy of the automatically generated candidate TM [9]. Next is a review of tagging on which our improvement of analyst-tool interaction is based.

B. Tagging in Social Computing and Software Engineering

The concept of tagging, as it is currently used in software engineering, comes from the social computing domain. Treude and Storey [18] provided a definition that is suitable for our research: A *tag* is a freely chosen short textual label that is associated with or assigned to a piece of information. A tagging system consists of three main components [19]: tag users, the objects being tagged, and the tag themselves.

Tagging in Web systems like Flickr and YouTube is often referred to as social bookmarking. The success is closely related to tagging's informal and bottom-up nature: Tags do not have to be predefined, every user can choose their own tags, and the number of tags per information item is arbitrary [18]. A fundamental characteristic of tags is that coherent and rich categorization schemes emerge even though unsupervised tagging is performed by individual users who can define their own vocabulary. In another word, it is the power-law-like long-tail shape of tag distribution that makes socially exchanging tags beneficial to users [19, 20, 21].

In software engineering, tags have been used for decades for annotating check-in and branching events in version control systems, as well as for documenting issues in bug tracking systems. Comments in source code can also be seen as tagging, though the purpose is primarily for explaining program functionality. Storey *et al.* [22] showed that comments could be turned into waypoints for code exploration, and their prototype tool — Tags for Software Engineering Activities (TagSEA) implemented as an Eclipse plug-in [23] — was shown to provide the user-defined navigation structures that are lacking in traditional code annotations.

Not only can tags help individual developers find their ways in the code base, tags also facilitate collaborations. Treude and Storey [18] reported industrial tagging practices within IBM's Jazz, a collaborative software development platform. Their study showed that Jazz's tagging mechanism was eagerly adopted by practitioners and the tags were used mainly to support communicating concerns and coordinating development tasks.

As social tagging has been only recently adopted, the smaller participation by the members of the software engineering community (e.g., an average of 233 tagging instances occurred over 3 TagSEA-adopted projects in about 2 years [23]) sets apart the kind of tagging done in software development from the tagging performed in Flickr and YouTube. The situation could be improved by automated ways of recommending tags from Q&A sites [24] or generating natural language code summaries [25, 26]. As far as manual tagging is concerned, Biegel *et al.* [27] proposed a gamification approach to motivate developers to tag more code fragments (i.e., Java methods) with more socially converging tags.

Tagging in tracing was previously investigated by Hale *et al.* [12]. The tagging feature was implemented in a courseware so that the student team members could tag various software artifacts, e.g., development ideas, code commits, project wikis, etc. These tags, then, could be used potentially as lightweight traceability links to support ongoing and post project analyses. While the work by Hale *et al.* [12] tried to use the tags as a back-end content mapping (i.e., TM generation) mechanism, the focus of our research is on exploring tagging as a front-end feature and how such an interface design might affect analysts' vetting and finalizing of the TMs in assisted tracing.

III. Research Questions

We derive our research questions based on the prior work reviewed in the previous section. Our first research question is concerned with the nature of the tags produced in assisted tracing. The literature suggests a power-law-like distribution of tag instances in both social computing [19] and software engineering [18]. In fact, the phenomenon that software follows power laws is shown to be pervasive, appearing at various levels of abstraction and in diverse systems and languages [28]. Formally, a power law is a probability distribution function in which the probability that a random variable takes a value is proportional to a negative power of that value:

$$P(X = x) \propto cx^{-k} \quad \text{where} \;\; c > 0, k > 0 \,. \qquad (1)$$

The direct plot of equation (1) results in a long, fat tail.[1] This shape of distribution indicates that a smaller percentage, $(a\times 100)\%$, of some input is generally responsible for a greater percentage, $(b \times 100)\%$, of some result [28]. A special case is the Pareto principle where a=0.20 and b=0.80. Boehm's observation [29] — "80% of the contribution comes from 20% of the contributors" — is among the popular instances of the Pareto principle in software engineering. In a power law's general form, however, there is no need for a=0.20 and b=0.80, nor is there a need for $a+b$=1. While power laws have important implications for practice [28], research on tagging in software engineering [12, 18, 22, 23, 24, 27] has not yet tested this property rigorously. To address the gap, we set out to examine

$\mathbf{RQ_1}$: To what extent do the tags produced in assisted tracing follow power laws?

As pointed out by Cuddeback *et al.* [8], the overall quality of assisted tracing is determined by the *accuracy* of the final TM submitted by the human analyst. Therefore, we are interested in assessing

$\mathbf{RQ_2}$: To what extent do the tags enhance the accuracy of analyst-submitted final TMs?

While our first research question focuses on tag *appearance* and second on tag *usage*, our last research question investigates the *interaction* between the two.

$\mathbf{RQ_3}$: How do analysts use the produced tags in assisted tracing?

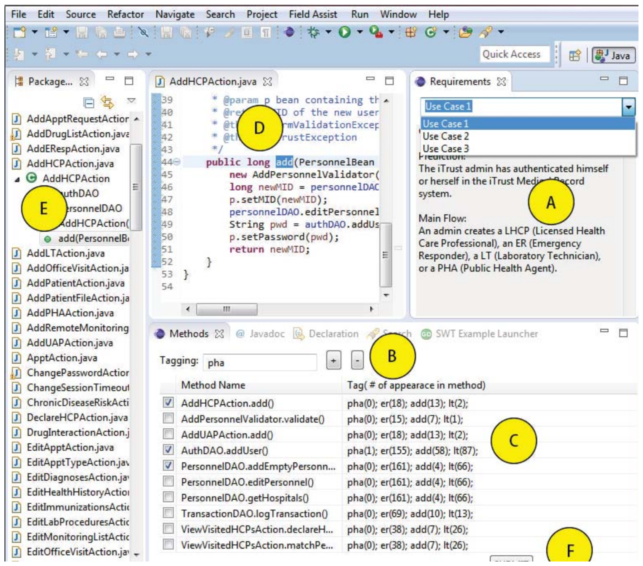
IV. Methodology

A. Research Setting

We developed a prototype tool for assisted tracing in our research. Fig. 1 shows a screenshot of the tool, which implements tagging as a front-end feature. Compared with standalone tracing tools like RETRO [2] and Poirot [3], our tool is implemented as an Eclipse plug-in so that the analyst can take full advantage of the integrated development environment (IDE) to support tracing, e.g., by executing or debugging the software to better locate traceability links. As mentioned earlier, traceability is not an end in itself. By having the TM embedded inside the IDE, we intend to provide the analyst a more seamless way to use the traceability information to support software engineering tasks, such as verification and validation, regression testing, and so forth.

Our tool design is informed by the basic features of RETRO [2] and Poirot [3], as well as by our own experience from building an assisted tracing tool to understand how the information environment affects analyst's behavior [10]. Two key aspects — embedded in an IDE and allowed for explicit tagging — set our current tool shown in Fig. 1 apart from other

[1]Plotted in logarithmic scale, a power law traces a straight line whose slope is k.



```
    * @param p bean containing th
    * @re    ID of the new user
    * @th    rmValidationExcep
    * @th    rustException
    */
   public long add(PersonnelBean
      new AddPersonnelValidator(
      long newMID = personnelDAC
      p.setMID(newMID);
      personnelDAO.editPersonnel
      String pwd = authDAO.addUs
      p.setPassword(pwd);
      return newMID;
   }
}
```

Prediction:
The iTrust admin has authenticated himself or herself in the iTrust Medi   ecord system.

Main Flow:
An admin creates a LHCP (Licensed Health Care Professional), an ER (Emergency Responder), a LT (Laboratory Technician), or a PHA (Public Health Agent).

Tagging: pha

	Method Name	Tag(# of appearace in method)
✓	AddHCPAction.add()	pha(0); er(18); add(13); lt(2);
	AddPersonnelValidator.validate()	pha(0); er(15); add(7); lt(1);
	AddUAPAction.add()	pha(0); er(18); add(13); lt(2);
✓	AuthDAO.addUser()	pha(1); er(155); add(58); lt(87);
✓	PersonnelDAO.addEmptyPersonn...	pha(0); er(161); add(4); lt(66);
	PersonnelDAO.editPersonnel()	pha(0); er(161); add(4); lt(66);
	PersonnelDAO.getHospitals()	pha(0); er(161); add(4); lt(66);
	TransactionDAO.logTransaction()	pha(0); er(69); add(10); lt(13);
	ViewVisitedHCPsAction.declareH...	pha(0); er(38); add(7); lt(26);
	ViewVisitedHCPsAction.matchPe...	pha(0); er(38); add(7); lt(26);

Fig. 1. Screenshot of the assisted tracing tool used in our work. The tool is implemented as an Eclipse plug-in that delivers tagging as a front-end feature.

state-of-the-art tracing tools. We next describe some important tool features as they relate to assisted tracing.

The tool displays the artifacts to be traced in its top-right corner. In Fig. 1-A, the drop-down menu allows the analyst to select a use case (UC) to view and trace. The tagging interface is shown in Fig. 1-B where the analyst can freely enter a keyword and then click "+" or "−" to add the keyword into or remove the keyword from the tracing of the selected UC. In our current implementation, adding an already existent keyword has no effect, and so does removing a non-existing keyword. In the future, we plan to enhance tagging by incorporating and updating keyword's weight. Fig. 1-C displays the candidate TM by listing the name of the method potentially implementing the selected UC. Once a tag is entered, our tool computes the number of tag appearance (case insensitive) inside each method and displays this information alongside the method. This tag-appearance cue can assist the analyst's vetting of the candidate TM. Double clicking a method in Fig. 1-C triggers the full method body to be shown in Fig. 1-D, as well as the synchronization of the method's view inside Eclipse's package explorer as shown in Fig. 1-E. The analyst selects or deselects the links of the candidate TM in Fig. 1-C, and then presses "SUBMIT" of Fig. 1-F to finalize the TM.

We adopted the iTrust dataset[2] in our study. 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 [30].

[2]http://agile.csc.ncsu.edu/iTrust

TABLE I
REQUIREMENTS TRACES USED IN THE EXPERIMENT

ID (our study)	Title (iTrust ID)	|true links|
UC1	Create Personnel (UC-2)	4
UC2	Find LHCPs (UC-29)	8
UC3	Identify Risk of Chronic Disease (UC-16)	7

TABLE II
PARTICIPANT BACKGROUND

	Working Experience (in years)				Most Familiar Programming Languages			
	No	$\leq$1	1∼5	$\geq$5	Java	C/C++	C#	Others
# of part.	8	2	17	1	23	19	5	12

Our study used iTrust's version 18.0 which was released in Fall 2014 and defined non-empty traceability links for 38 UCs at the Java method level. We randomly chose 3 UCs and treated the TMs defined in the project repository as the answer set. Table I lists these requirements tracing tasks. Because our primary goal is to study analyst's tagging during tracing, we restrict the size of each UC's candidate TM to be 10. Fig. 1-C illustrates that only 10 methods are displayed for analyst to interact with. To not imposing any displaying order, the methods are sorted alphabetically. While such design decisions pose important limitations discussed later in Section V-D, we were able to control the same information patch for analysts to pursue the correct traceability links [10]. Consequently, we could focus our analysis on analyst's tagging behavior and ignore controlled factors like patch profitability — 100% recall and varying precision for 3 UCs (cf. Table I).

We recruited 28 upper-division students in computer science and software engineering from our institutes. As the prior work shows that grade level, software engineering experience, tracing experience, and tracing location had no effect on analyst performance [7], we make no further distinction among our participants. Table II summarizes participant background. None of the participants knew iTrust before the experiment, but most of them reported being familiar with Java or another object-oriented language. Python and SQL were among other popular languages that our participants were familiar with.

During the experiment, each participant (analyst) worked alone in a lab. The analyst first filled out a pre-study survey of their background, and then learned how to use the assisted tracing tool. They could execute iTrust in Eclipse if they wanted and were free to use tagging during tracing. The analyst performed requirements-to-source-code tracing in the context of ensuring the satisfaction of iTrust's critical requirements (i.e., UC1, UC2, and UC3). Specifically, we asked the analyst to find all and only the methods implementing a particular UC, but the analyst could trace the UCs in any order they would prefer. A constraint was that the analyst needed to use only the provided tracing tool and not to use internet or any other resources in the experiment. For each participant's experiment session, one researcher was present to run the tracing tool 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 participant's session lasted approximately 30 minutes.

B. Data Collection and Analysis

Our methodology follows a mixed approach to collect both quantitative and qualitative data. In order to gather quantitative data on the use of tags in assisted tracing, our tool logged the tag keywords appeared in each participant's session. Qualitative data were collected through video screen capturing of all the tracing sessions by Camtasia, ethnographic-style observations made by the researchers, and exit-interview notes followed by questions and clarifications.

While the data on tag appearance can be reported in a straightforward fashion, tag usage data need additional analysis. We list below a representative example for each of the three cases in our analysis to illustrate the determination of what tag was used to finalize which method.

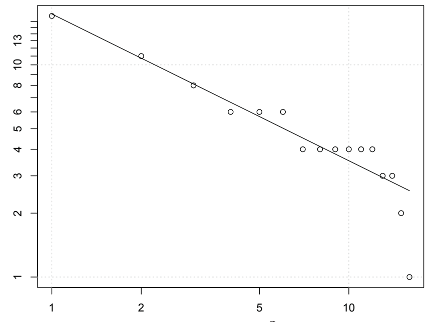
- When tracing UC1, analyst X entered "professional" as a tag keyword. However, she scanned the appearance of this tag and entered another tag to continue tracing. In this case, tag "professional" was linked to no method in analyst X's tracing of UC1.
- The new tag that analyst X entered was "ER" (short for "Emergency Responder" in UC1). This time, she noticed that "er" appeared 155 times in AuthDAO.addUser(). This information, together with the method's comments, helped analyst X to include AuthDAO.addUser() in her final TM of UC1. Therefore, we associated "er" with AuthDAO.addUser() for analyst X.
- Things were not always as black and white as the above. Analyst Y tagged "zip" in his tracing of UC2. He also stated explicitly that "*zip is promising*" in vetting PersonnelDAO.getPersonnel(). However, we could not find firm evidence that he finalized PersonnelDAO.getPersonnel() because of the tag. As a result, we marked no relationship between "zip" and PersonnelDAO.getPersonnel() or any other method in analyst Y's tracing of UC2.

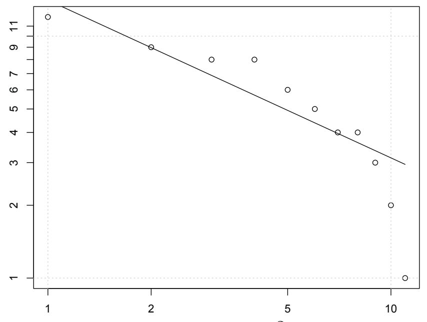
Two researchers applied our scheme to independently code all the tag usage data. Due to the conservative nature of our scheme, i.e., relating tag and method positively only if firm evidence was present, the two researchers achieved an over 90% agreement on coding all tracing sessions. The remaining disagreements were resolved collaboratively and quickly, benefiting from the involvement of a third researcher, and once again, from our coding scheme's conservativeness.

V. RESULTS

A. The "What": Tag Appearance

We examined $\mathbf{RQ_1}$ by plotting the tag appearance data of each tracing task in logarithmic scale. Fig. 2 shows the results. Our depiction of power laws follows a cumulative definition [28] homomorphic to equation (1). Take UC1's plot in Fig. 2 as an example, the top-left node indicates that 17 tags are being tagged by at least 1 analyst, and the bottom-right node implies there is 1 tag receiving tagging from more than 16 analysts.


UC1 (k=1.69, r^2=0.97)


UC2 (k=1.65, r^2=0.84)

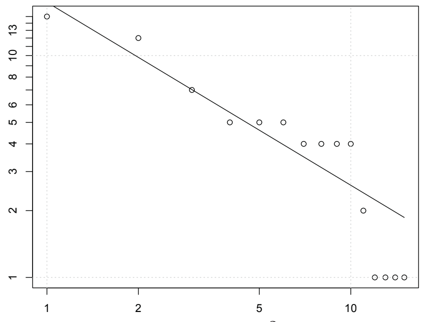

UC3 (k=1.82, r^2=0.91)

Fig. 2. Tag appearance data plotted in logarithmic scale where x-axis represents the number of tag users (analysts) and y-axis represents the number of tags.

TABLE III
TOP-FIVE, CASE-INSENSITIVE TAGS OF EACH TRACING TASK

UC1	UC2	UC3
lhcp	patient	risk
er	lhcp	heart
lt	diagnose	diabetes
pha	zip	patient
add	history	disease

When fitting the tag appearance data to a line, we obtained r^2 value reaching 0.84 or higher for all 3 UCs. Fig. 2 also shows each UC's fitted regression line. The k value ranges from 1.65 to 1.82, which is homogeneous and in line with the k's reflected in software power laws [28]. Thus, for $\mathbf{RQ_1}$, our results indicate a good fit of the analyst-produced tags and power laws.

It is important to point out that the tagging mechanism in our tool was readily adopted by the analysts, who encountered no difficulty in learning the tagging feature and applied tagging seamlessly throughout the tracing tasks. This is not surprising as the prior research already showed tagging's eager adoption by the software developers [18].

What turns out to be surprising is that the tags produced in our study were predominantly drawn from the requirements descriptions, as listed in Table III. This is in stark contrast to code tagging where Biegel *et al.* [27] found that most participants in their study felt uncomfortable in just copying words from the code when tagging Java methods. The difference is fundamental: While the objects being tagged are homogenous in contemporary software engineering approaches (e.g., work items in [18] and code fragments in [27]), tracing can and usually does involve heterogeneous objects. As a result, it is the traceability relationships, and not the objects in isolation, that are subject to tagging. The excessive use of requirements terms as tag keywords in our study could reflect the influence exerted by the directionality of the trace links. Future work can investigate how tagging may differ in other tracing tasks, e.g., vetting source-code-to-requirements links for validation and assurance purposes.

B. The "How": Tag Usage

We present summary statistics about tag usage in Table IV. Statistics reported in Table IV are mean values. Averagely speaking, analysts used more than half of the tags to support TM finalization. About a fourth to a third of the tags, however, were unused according to our data analysis (cf. Section IV-B). On the method side, over two thirds were traced with the help of tags whereas some small portions were not.

To answer $\mathbf{RQ_2}$, we further computed recall, precision, and F_2 [13] for the analyst-submitted links that were traced with tagging support and those that were finalized without tagging. The rightmost columns of Table IV show these results. For all 3 tracing tasks, tagging made a statistically significant difference on precision (Wilcoxon Signed Ranks test applied over repeated measurements, α=.05). Such a significant effect was not detected on recall or F_2. We posit the main reason why tagging greatly improved precision is that it allowed analysts to look for specific keywords to accept plausible links. This "accept-focused" strategy is found to be effective, though combining it with other strategies (e.g., "first good link") can result in even more accurate final TMs [1].

C. The "What" Meets The "How"

Addressing $\mathbf{RQ_3}$ required us to interconnect the power-law appearance of tags with their usage. Fig. 3 presents our analysis results. For each UC, we identified the most appeared tags (i.e., head of long tail) by calculating the values of a and b. For instance, 29% of the UC1 tags cumulated 72% of appearance. For the tail, we distinguished the tags unique to an analyst from the remaining tags. A tag is unique if only a particular analyst used it for a given tracing task.

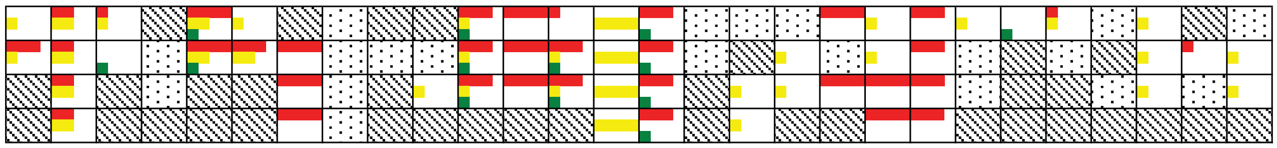
An important finding of Fig. 3 is that all the true links were able to be traced via tagging support. Given the analysts in our study performed tracing individually, Fig. 3 also shows that exchanging the tags among the analysts could facilitate the sharing of useful cues in tracing, thereby improving analyst performance. We next present several major tag usage patterns to illustrate tagging's positive and negative influences.

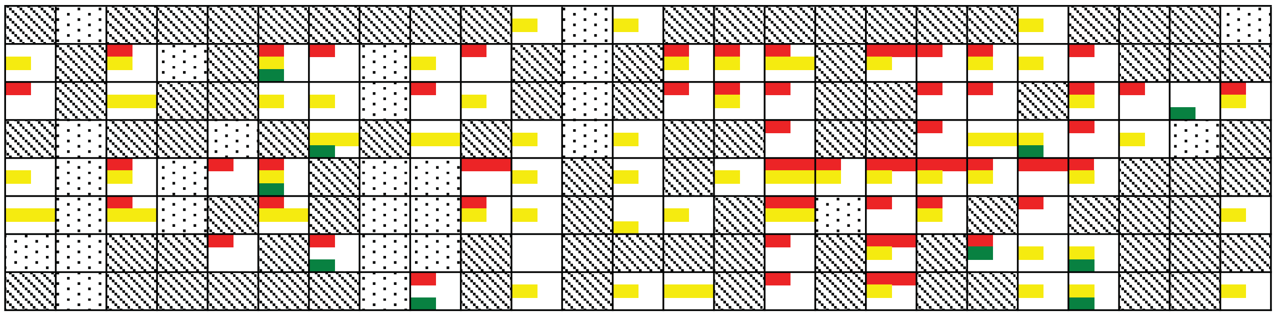
Directly confirming. In tracing UC3, "risk" was the most used tag and served as a starting point for many analysts' vetting. The method HeartDiseaseRisks.getDiseaseRiskFactors(), which is a true link, contains 20 "risk" instances. Furthermore, only this method contains "heart" which many analysts tagged as well. Using both tags allowed for the correct selection of the true link in the final TM. This example also shows that tags are not isolated but interrelated.

Misleading. Using the "risk" tag did not always lead to correct decisions. RiskDAO.getFamilyHistory(), which is a false link,

TABLE IV
DESCRIPTIVE STATISTICS (MEAN VALUES) RELEVANT TO TAG USAGE AND ACCURACY OF ANALYST-SUBMITTED FINAL TMS

Tracing task	Tag (proportion and absolute size)				Method (proportion and absolute size)				Recall (tagging)		Precision (tagging)		F_2 (tagging)	
	used for tracing		unused for tracing		traced via tag		traced not via tag		with	w/o	with	w/o	with	w/o
UC1	67.1%	2.76	32.9%	1.35	80.0%	3.45	20.0%	0.85	.69	.34	.76	.52	.68	.39
UC2	73.9%	3.56	26.1%	0.89	69.2%	3.85	30.8%	1.80	.52	.52	.84	.64	.55	.59
UC3	66.5%	3.39	33.5%	1.06	71.7%	3.85	28.3%	1.55	.74	.56	.85	.70	.75	.64


UC1 (4 true links/methods, 28 analysts, unit cell length = 4 tags, $a = 0.29$, $b = 0.72$)


UC2 (8 true links/methods, 25 analysts, unit cell length = 2 tags, $a = 0.38$, $b = 0.71$)

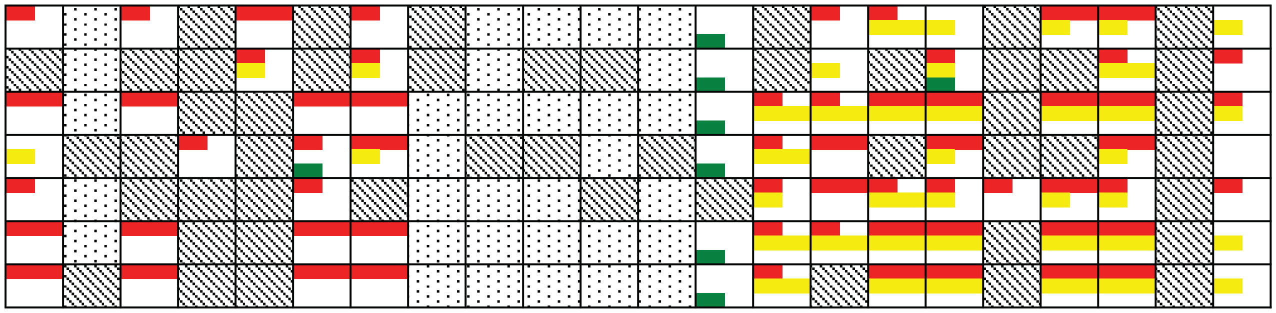

UC3 (7 true links/methods, 22 analysts, unit cell length = 2 tags, $a = 0.33$, $b = 0.73$)

Fig. 3. Linking tag appearance and usage in tracing. For each tracing task, rows represent true links (i.e., Java methods that should be traced to the UC), and columns represent analysts who completed the UC tracing. A white-background cell implies that the analyst used tagging to select the method in the submitted final TM. Tags are of 3 types: most appeared (🟥), unique to the analyst (🟩), and the rest (🟨). Tag length corresponds to size. The dotted cell indicates that the analyst selected the method, but not via tagging. The striped cell means that the analysis did not select the method in the final TM.

has one of the most "risk" appearances in it. Many analysts included this false link in their final UC3 TMs. Therefore relying only on a single tag can be insufficient to decide a candidate traceability link's relevance.

Combining with other cues. Besides the use of multiple tags mentioned above, analysts took advantage of other types of cues in tracing. A unique tag, "personnel", helped one analyst to make the correct decision of selecting PersonnelDAO.addEmptyPersonnel() in the final TM. The analyst stated that not only "personnel" appeared in the method, but it appeared in the class name and the method name. We thus conjecture that other cues in the tracing environment may be exploited to update tag weight and utility.

Unused. As shown in Table III, many analysts tagged "zip" in UC2's tracing. Due to nonappearance in the methods, the tag was abandoned quickly. Our analysis shows that a true link PatientDAO.getPatient() invokes PatientLoader.loadCommon() in which "zip" appears. Unfortunately, none of the analysts followed the invocation relations in confirming PatientDAO.getPatient(). We can identify an opportunity of propagating the tag (e.g., through method invocations) so that more relevant links can be brought to analyst's attention.

D. Study Limitations

A major limitation is our control over what candidate traceability links were displayed and how they were displayed. We achieved 100% potential recall [1] in all the 3 tracing tasks; however, the alphabetical ordering could make the tool look too artificial and even not trustworthy. In this sense, analysts may be more motivated to vet the tool's output and to do so via the tagging support. Given that analysts did cast doubt on tool's output [17], we feel that front-end features like tagging could be particularly valuable for providing cues so that analysts may understand better *why* the tool performed in certain ways.

Our work with student participants limits how the results could be generalized. Egyed *et al.* [31] note that in many industrial settings people have no intimate system knowledge during trace recovery. Prior studies have also used students with low levels of industry experience to represent new people joining a company [1, 5, 6, 7, 31]. Nevertheless, it would be interesting to study how familiarity levels may alter the tracing and tagging behaviors. Similarly, using multiple datasets with more diverse tracing tasks helps tackle important threats to external validity.

VI. Conclusions

Research on assisted tracing has highlighted the human-centered nature of software traceability. The main contributions of this paper are the integration of tagging as a front-end feature into a tracing tool, and the examination of how human analysts interact with such a feature. Our results show that analysts adopt well the tagging-to-tracing practice, uncover tags' power-law-like distribution, and reveal the positive effect of tagging on the precision of analyst-submitted final TMs.

Our future work includes conducting in-depth studies (e.g., with industrial or original developers, incorporating hard-to-retrieve traces [32], etc.) to lend strength to the preliminary results reported here, enriching the tagging feature (e.g., allowing tag weight to be updated), connecting front-end tags with back-end trace retrieval algorithms (e.g., via user feedback [33]), and exploring ways to socially exchange tags so as to improve analyst performance in assisted tracing.

Acknowledgement

We thank all of our study participants. The work is funded in part by the U.S. National Science Foundation (CCF-1350487), the National Natural Science Foundation of China (No. 61272169, 61472034), Program for New Century Excellent Talents in University (No. NCET-13-0041), and Beijing Higher Education Young Elite Teacher Project (No. YETP1183).

References

[1] W.-K. Kong, J. H. Hayes, A. Dekhtyar, and O. Dekhtyar, "Process improvement for traceability: a study of human fallibility," in *RE*, 2012, pp. 31–40.

[2] 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.

[3] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova, "Best practices for automated traceability," *IEEE Computer*, vol. 40, no. 6, pp. 27–35, June 2007.

[4] S. Klock, M. Gethers, B. Dit, and D. Poshyvanyk, "Traceclipse: an Eclipse plug-in for traceability link recovery and management," in *TEFSE*, 2011, pp. 24–30.

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

[6] 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.

[7] 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.

[8] 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.

[9] A. Dekhtyar and J. H. Hayes, "Studying the role of humans in the traceability loop," in *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds. Springer, 2012, pp. 241–261.

[10] N. Niu, A. Mahmoud, Z. Chen, and G. Bradshaw, "Departures from optimality: understanding human analyst's information foraging in assisted requirements tracing," in *ICSE*, 2013, pp. 572–581.

[11] C. Treude and M.-A. D. Storey, "How tagging helps bridge the gap between social and technical aspects in software development," in *ICSE*, 2009, pp. 12–22.

[12] M. Hale, N. Jorgenson, and R. Gamble, "Analyzing the role of tags as lightweight traceability links," in *TEFSE*, 2011, pp. 71–74.

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

[14] 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.

[15] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "On integrating orthogonal information retrieval methods to improve traceability recovery," in *ICSM*, 2011, pp. 133–142.

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

[17] 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.

[18] C. Treude and M.-A. D. Storey, "Work item tagging: communicating concerns in collaborative software development," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 19–34, January/February 2012.

[19] V. Robu, H. Halpin, and H. Shepherd, "Emergence of consensus and shared vocabularies in collaborative tagging systems," *ACM Transactions on the Web*, vol. 3, no. 4, article 14, September 2009.

[20] S. Sen, S. K. Lam, A. M. Rashid, D. Cosley, D. Frankowski, J. Osterhouse, F. M. Harper, and J. Riedl, "Tagging, communities, vocabulary, evolution," in *CSCW*, 2006, pp. 181–190.

[21] M. Ames and M. Naaman, "Why we tag: motivations for annotation in mobile and online media," in *CHI*, 2007, pp. 971–980.

[22] M.-A. D. Storey, L.-T. Cheng, J. Singer, M. J. Muller, D. Myers, and J. Ryall, "How programmers can turn comments into waypoints for code navigation," in *ICSM*, 2007, pp. 265–274.

[23] M.-A. D. Storey, J. Ryall, J. Singer, D. Myers, L.-T. Cheng, and M. J. Muller, "How software developers use tagging to support reminding and refinding," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 470–483, July/August 2009.

[24] X. Xia, D. Lo, X. Wang, and B. Zhou, "Tag recommendation in software information sites," in *MSR*, 2013, pp. 287–296.

[25] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. L. Pollock, and K. Vijay-Shanker, "Automatic generation of natural language summaries for Java classes," in *ICPC*, 2013, pp. 23–32.

[26] L. F. Cortés-Coy, M. L. Vásquez, J. Aponte, and D. Poshyvanyk, "On automatically generating commit messages via summarization of source code changes," in *SCAM*, 2014, pp. 275–284.

[27] B. Biegel, F. Beck, B. Lesch, and S. Diehl, "Code tagging as a social game," in *ICSME*, 2014, pp. 411–415.

[28] P. Louridas, D. Spinellis, and V. Vlachos, "Power laws in software," *ACM Transactions on Software Engineering and Methodology*, vol. 18, no. 1, article 2, September 2008.

[29] B. W. Boehm, "Industrial software metrics: a top-ten list," *IEEE Software*, vol. 4, no. 9, pp. 84–85, September 1987.

[30] A. Meneely, B. Smith, and L. Williams, "iTrust electronic health care system case study," in *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds. Springer, 2012, pp. 425–438.

[31] 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.

[32] M. Gibiec, A. Czauderna, and J. Cleland-Huang, "Towards mining replacement queries for hard-to-retrieve traces," in *ASE*, 2010, pp. 245–254.

[33] Y. Shin and J. Cleland-Huang, "A comparative evaluation of two user feedback techniques for requirements trace retrieval," in *SAC*, 2012, pp. 1069–1074.