

Gray Links in the Use of Requirements Traceability

Nan Niu
Department of EECS
University of Cincinnati
Cincinnati, OH, USA 45221
nan.niu@uc.edu

Wentao Wang
Department of EECS
University of Cincinnati
Cincinnati, OH, USA 45221
wang2wt@mail.uc.edu

Arushi Gupta
Department of EECS
University of Cincinnati
Cincinnati, OH, USA 45221
gupta2ai@mail.uc.edu

ABSTRACT

The value of traceability is in its use. How do different software engineering tasks affect the tracing of the same requirement? In this paper, we answer the question via an empirical study where we explicitly assign the participants into 3 trace-usage groups of one requirement: finding its implementation for verification and validation purpose, changing it within the original software system, and reusing it toward another application. The results uncover what we call “gray links”—around 20% of the total traces are voted to be true links with respect to only one task but not the others. We provide a mechanism to identify such gray links and discuss how they can be leveraged to advance the research and practice of value-based requirements traceability.

CCS Concepts

•Software and its engineering → Requirements analysis; Traceability;

Keywords

Traceability, using traceability, software engineering task, requirements change, requirements reuse, gray links

1. INTRODUCTION

Attaining and maintaining requirements traceability place a significant burden on software engineers. Low adoption of traceability in practice, since reported in Gotel and Finkelstein’s seminal work [24], has been notoriously persistent. There are tools helping automate the generation of traceability links, most notably via information retrieval (IR) algorithms. However, these tools are far from outputting all the correct links (100% recall [38]) and only the correct links (100% precision [38]).

To improve recall and precision, various proposals are made. Some combine IR’s textual cues with other sources of information: structural [40], semantic [36], historical [1],

etc. Some manipulate the source and the target of traceability, e.g., configuring query [43] and expanding corpora [10]. Some act on the retrieved link candidates: pruning them to reduce irrelevance [27, 57], clustering them to arrange for natural groupings [13, 46], and bundling them to exploit partial knowledge [14, 21]. Yet, many more methods have been proposed in the past decade [5].

What does a human analyst do when presented with the output from the automated requirements tracing tool? The seminal work by Hayes and Dekhtyar [26], together with their series of studies [8, 11, 31, 32], showed that nobody would take the tool output as their finally-approved, “certified” trace links. Such an invariably-overriding behavior by the human analysts highlights that traceability “cannot live without them” [26]. However, it “cannot live with them” either, because the analysts’ overriding was often harmful [26]. Ironically, the most consistently and greatly degraded trace links were the high-recall and high-precision ones produced by the tool [9].

Thus, a tool that induces better analysts’ performance in finalizing their trace links is considered to be valuable. Such judging-the-tool-output is referred to as *vetting*, where an analyst is asked to browse and verify the automatically generated trace links [9]. However, without knowing how the traceability would actually be *used*, analysts waste much time and encounter much uncertainty during vetting [31]. Traceability is not an end in itself but a means to support various software engineering tasks [6]. Surprisingly, little is known about the impact of these tasks on analysts’ judgment of the trace links.

In this paper, we shorten the gap by exploring the impact of use tasks on traceability. For the same requirement, we explicitly assign the human participants with three tasks: vetting its traceability in a broad verification and validation (V&V) context, changing it within the original system, or reusing it toward another system. While the V&V task involves no direct use of the requirements trace links, the other two tasks do in that the participants are asked to use their chosen trace links to carry out the specific change or reuse task being assigned to them *a priori*.

The results, to our surprise, do not support the conjecture that analysts knowing how the trace links are to be used will make better vetting decisions [31]. For requirements change and reuse tasks, an in-depth analysis uncovers that around 20% of the total traces are voted as true links with respect to one task but not to both. We call them “gray links” and believe their existence directly shakes some contemporary bases of traceability, such as assessing a tracing method per-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

FSE’16, November 13–18, 2016, Seattle, WA, USA
© 2016 ACM. 978-1-4503-4218-6/16/11...\$15.00
<http://dx.doi.org/10.1145/2950290.2950354>

formance against a single answer set and predefining a fixed information model for all traceability uses.

The contributions of this paper lie in the discovery of use-task-induced gray links and in the analysis of how the gray links can be exploited to better support the use tasks. In what follows, we review related work in Section 2 and present our empirical study design in Section 3. The experimental results are analyzed in Section 4 where a mechanism of identifying the gray links is introduced. Section 5 examines the use of the gray links. Section 6 discusses the threats to validity as well as the implications of our work. Section 7 concludes the paper.

2. BACKGROUND AND RELATED WORK

2.1 Traceability and Its Automation

The need to describe and follow the life of a requirement from its origin to its realization gave rise to *requirements traceability* [24], which can be generalized to other kinds of artifacts like a design model, a piece of code, and a test case. Traceability, then, is about “connecting the dots” among software artifacts to answer questions of both the software product and its development process [6].

Because it is tedious and error-prone to create and maintain the traceability information manually, automated support is developed and much is based on information retrieval (IR). These methods facilitate the recovery of the candidate traceability links¹ by relying on the textual descriptions of the query requirement and the target artifacts. Researchers have applied many IR methods in automated tracing [2, 36, 37, 39, 56, 57]. Extensive empirical evaluation results show that these methods have a comparable level of effectiveness [52], where effectiveness is measured primarily based on the two IR metrics: recall and precision [38].

The prerequisite for computing recall and precision is the known “answer set”, as shown in Figure 1. For a given requirement, its answer set specifies all the correct trace links and only those links. Therefore, a high recall value indicates a more complete coverage of the correct links by the trace retrieval method, whereas a high precision value signals a low proportion of noise contained in the candidate links.

IR-based requirements tracing methods typically return the candidate links with high-recall but low-precision values [27]. In most cases, a recall of 90% is achievable at precision levels of 5-30% [1, 5, 27, 36, 37, 39, 52, 57]. The high recall value can be attributed to the size of the tracing targets: A medium-sized code base, for example, may contain hundreds and thousands of artifacts (e.g., classes or methods) and an after-the-fact² trace retrieval method that returns all the code artifacts is guaranteed to have a 100% recall. In contrast, some IR applications like Web search must handle millions or billions of retrievables³, making a high-recall performance practically infeasible.

A root cause of the trace retrieval methods’ low precision is the length of the query requirement. Each trace query can

¹The traceability links are “candidate” until an analyst vets them; the “final” traceability links are the ones approved [8].

²After-the-fact tracing operates on a snapshot of the software system by *not* considering the evolution of the tracing source and the tracing target over time [12].

³As an example, the estimated size of Web pages indexed by Google reached nearly 50 billion in February 2016 (<http://www.worldwidewebsize.com>).

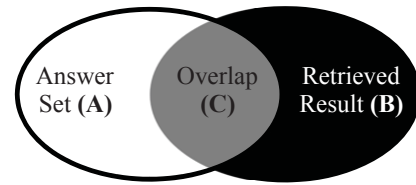


Figure 1: $\text{Recall} = \frac{|C|}{|A|}$ and $\text{Precision} = \frac{|C|}{|B|}$ are set-based measures relying on the known “answer set”.

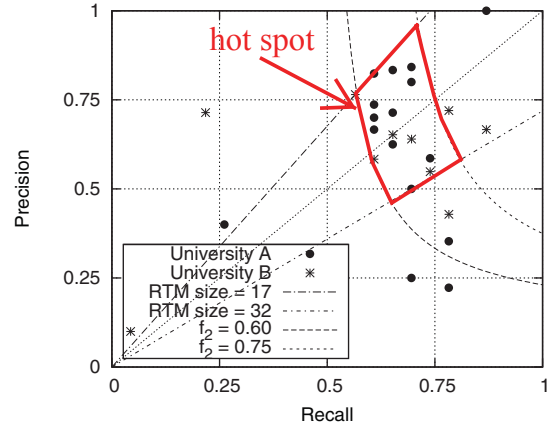


Figure 2: “Hot spot” in the recall-precision space where the analysts’ submitted trace links are observed to concentrate (adopted from [8]).

be a sentence (e.g., a feature description) or several sentences (e.g., a user story or a use case) written in natural language. Compared to the short Web-search queries⁴, long queries can cause irrelevant links to be retrieved, decreasing precision.

In short, IR-based methods automate the after-the-fact recovery of candidate traceability links to a large extent. Assessed with the “answer set”, these methods cannot achieve a high coverage of all the correct links without also returning a great proportion of incorrect ones. Beyond recall and precision, additional metrics are proposed to take the human factors into consideration. For example, the mean average precision (MAP) provides a sense of the browsability of retrieved link candidates in the ranked list [57] and the F_2 measure is commonly adopted to combine together recall and precision because it is assumed that analysts are better at removing incorrect links than finding missing links [27].

2.2 Human in the Traceability Loop

How does a human analyst actually behave when browsing and verifying the candidate traceability links? Hayes and Dekhtyar [26] began the research thrust on human factors and repeatedly recognized analysts’ fallibility in the requirements tracing process [9]. Such fallibility, quite surprisingly, was observed to be predictable [9]. In Figure 2, for example, a “hot spot” of 26 student analysts’ finalized trace links could be depicted by the F_2 value and the size of the submitted requirements traceability matrix (RTM) [8].

To understand the factors causing analysts’ predictably fallible behavior, Dekhtyar *et al.* [11] examined 11 vetting

⁴An average length of 2.5 terms per query was used for retrieving the TREC HARD collection [30].

variables. Their examination revealed that only the accuracy of the initial candidate links and the analysts' effort expended in validating offered links had statistically significant effects on the final trace links, while the other 9 factors (e.g., tool used, software engineering experience, tracing experience, effort on searching for omitted links, etc.) did not make a difference. Complementary to observational studies, we explored the theoretical underpinning of analysts' requirements tracing behavior in light of the information foraging principles [48].

In sum, human factors fundamentally shape the landscape of automated traceability. As traceability decisions are human decisions in most situations, neglecting the analyst-tool interaction will only widen the gap between requirements tracing methods [27] and the analysts [8]. However, studies so far [8, 11, 31, 32], including our own [48, 60], have paid attention exclusively to analyst-tool reaction (i.e., vetting the links in after-the-fact tracing) rather than the interaction where analysts use the links to accomplish their tasks.

2.3 Value-based Traceability

The value of traceability is in its use. The key tenet of value-based software engineering is to shift the practice and research away from a value-neutral setting where every requirement, use case, object, test case, and defect is treated as equally important [28]. With value-based traceability, this means that not all trace links are the same [15]. Their values must be distinguished based on the use context.

"Who uses traceability" was answered by the early work of Ramesh and Jarke [54]. They identified a wide range of traceability practices with distinct low-end and high-end users of traceability. While low-end users were interested in solving specific tasks like allocating requirements to implementation units, high-end users tended to capture traces across product and process as a means of managing rationale. "When to use traceability" should not be constrained by a static set of artifacts in an after-the-fact and retrospective manner only. Prospective traceability tools, such as TRASE [2], ReCVisu+ [50], and TOOR [53], generate trace links *in situ* during artifacts' creation and modification over the project life cycle.

The most important factor, compared to the "who" and the "when", is the "why", i.e., the *purpose* of tracing [7]. Traceability is needed to meet various purposes, ranging from being mandated by standards like CMMI level 3 to supporting software engineering tasks like verification and validation (V&V) [27], software maintenance [34], product line reuse [51], concept location [47], software exploration [49], code foraging [45], hazard mitigation [35], etc. In fact, IR-based methods have been successfully applied in more than 20 different tasks [43]. However, the value of traceability is often assumed based on a purely technical perception rather than measured directly.

An exception is the experiment recently conducted by Mäder and Egyed [34] that studied the effect of requirements traceability on participants' software maintenance performance. The results show that participants provided with trace links performed on average 24% faster and created 50% more accurate solutions than the participants who performed those tasks without the provided links. These encouraging results help quantify the expected benefits of using traceability in concrete tasks. Building on these, our work tries to improve three aspects in the experimental de-

sign of [34]: having participants develop task solutions in the programming environment instead of sketching the solutions on paper, providing IR-based candidate links instead of perfect traces with 100% recall and 100% precision, and, most importantly, taking a value-based perspective instead of treating every trace equivalently.

Egyed and his colleagues pioneered the research on value-based traceability by introducing a language to capture the trace with uncertainties (e.g., code *c* "implements at least" requirement *r*) [14], adding to the language a set of constraints (e.g., *c* "cannot simultaneously be implementing and not implementing" *r*) [20], developing a scalable reasoning engine based on SAT solvers [21], and sharing the experiences of empirical studies [15, 16, 17]. Another important contribution is Lago *et al.*'s approach to customize traceability for stakeholders' activities at hand [33]. The key idea is to scope down all possible trace links to the essential ones codified in the core traceability paths. Although different activities require different core paths to tackle different traceability issues, the scoped approach is value-based: "By knowing in advance the use we aim at, we can consciously decide on which traceability links to document, and can exclude those that are not necessary" [33].

In a nutshell, requirements traceability must be not only measurably valuable [34] but also value-based [15]. Not all trace links should be equally important. Their varying importance hinges on their very purposes of existence, i.e., to support the activities that stakeholders must carry out [33]. Do different uses lead to different trace links, even for the same requirement? This is precisely the motivational question that drives our exploration of "gray links".

3. STUDY DESIGN

3.1 Terminology and Hypothesis

The term, "gray links", is not new. Kong *et al.* [31] used the term to refer to those trace links that analysts could not definitely judge as true or false links. The literature suggests two reasons for this phenomenon at an individual level.

- **Unknown or implicit guidance on final links' usage.** When analysts were asked to vet the candidate traceability links but unsure about how the links would be used, they spent a significant amount of time deliberating whether certain links should be approved [31]. However, little is known about how many of these troublesome, hard-to-decide links appear and how they may affect the actual usage.
- **Partial knowledge analysts have about the traceability.** A developer, for example, may remember that a given requirement is implemented in a subset of classes but not exactly which subset. A trace capture tool is proposed to allow such uncertainty to be expressed and exploited [14, 21].

Prior research thus revealed hard-to-judge links during trace capturing or vetting, i.e., judging "Yes, this is a link" or "No, this is not a link" [8, 9, 31] for the sake of creating the trace links for later use. Our focus in this paper is not on creating the traceability but on using it, or more accurately, on the interdependency of the two when the use task is defined explicitly and *a priori*. The hypothesis that we want to test empirically is what was posited by Kong *et al.* [31]:

H: Knowing how the final trace links are to be used will better equip the human to make the vetting decisions, i.e., to better judge “Yes, it is a link” or “No, it is not a link”.

Before detailing our experimental design in the next subsection, we clarify the terminology used in our work. A trace link expresses a relationship between a source artifact and a target artifact (e.g., “a method *implements* a requirement”). For a given link, we say it is *correct* if the expressed relationship agrees with some facts or reality. The answer set, typically defined by the original development team, documents all and only the correct traceability links⁵. It is against the correct links that an IR-based tracing tool’s output—and for that matter analysts’ vetting result—is compared. We regard a link as *true* if an analyst *judges* it to be correct. In another word, the true link is relative to an analyst’s judgment, whereas the correct link, intended to be authoritative, is not⁶. Finally, a link is *voted true* if a group as a whole favors the true judgment of the link. We present a mechanism for determining voted true links in Section 4 where our experimental results are analyzed.

3.2 Experimental Setup

We designed three conditions to test **H**, as illustrated in Figure 3. The control group (**G0**) received the “no direct use of the final trace links” treatment. Specifically, a V&V instruction was given for the human participants of **G0** to find all and only the correct implementation elements of a given requirement. For the same requirement, two experimental groups received different “final-trace use” treatments: We asked one group (**G1**) to change the requirement and the other (**G2**) to reuse it.

We chose iTrust⁷ to be our subject system. 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 serves as a traceability testbed for understanding the importance of security and privacy requirements in the healthcare domain [41]. iTrust has over 30 use cases (UCs) and 750 JSP/Java files. The answer set of correct requirements-to-source-code traceability links is defined by the project team in a spreadsheet, downloadable from the project’s Web site.

To test **H**, we selected one of iTrust’s requirements: UC11 (document office visit). This selection was informed by our evolutionary analyses of 15 iTrust releases [59]. The description of this requirement, together with that of the 3 tasks, is detailed in Table 1. Figure 4 provides further illustrations. The change task requires the modification of UC11 subflow [S4] so that the health care professional (HCP) is armed with the new capability to automatically check the interactions between the to-be-prescribed medications and the drugs currently taken by the patient (cf. Figure 4b).

For the requirements reuse task, we sketched an initial Java implementation of a student degree management sys-

⁵ Answer set may not be perfect (e.g., one contains outdated information [59]), but if the development team defines it, then it is intended to be authoritative in our opinion.

⁶ We realize our labelings of “correct” and “true” links may not be universally applicable (e.g., the “answer set” is sometimes called “ground truth”), but we believe the distinction is critical when human factors are considered in traceability.

⁷ <http://agile.csc.ncsu.edu/iTrust>

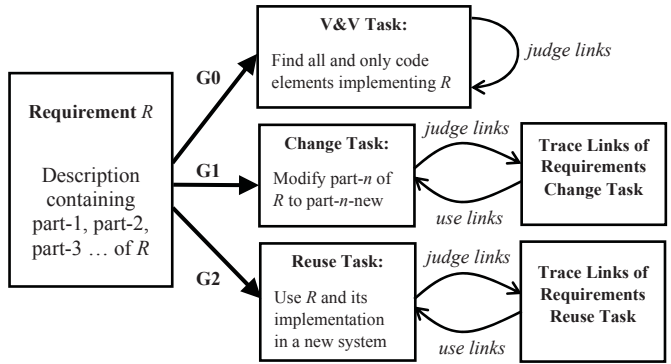


Figure 3: Different tasks on the same requirement.

tem (SDMS), which contains two hardcoded JSPs: one for identifying the student and the other for checking the degree requirements (cf. Figure 4c). The code base also defines 2 classes (Student.java and Course.java) without specifying any of their attributes or methods. The main purpose of our initial implementation is to provide some basic building blocks [29] for the participants to better comprehend the requirements reuse task. Only the hardcoded Figure 4c without the “checking data format” annotation was shown to the participants. Even though the data format checking ([S2] of iTrust’s UC11) was a good reuse candidate, the participants were instructed to reuse “iTrust’s UC11 and its implementation” in any way they deemed appropriate.

We developed a prototype tool based on Eclipse plug-in architecture. The tool development was built on our experience with the instrumentation of studying analysts’ information foraging and tagging in requirements tracing [48, 60]. We wanted the traceability tool to be seamlessly integrated with the programming environment because **G1** and **G2** participants would modify the code by using the trace links. We also wanted to impose a “vetting-before-using-the-links” order for **G1** and **G2** participants because otherwise the vetting decisions might not be recorded. For these reasons, we built the prototype tool instead of using the state-of-the-art tools like TRASE [2] in our experiment.

Three tool variants were devised for each of the **G0**, **G1**, and **G2** groups. Figure 5 illustrates our tool with the **G1** task. The requirements description and the task description are displayed in Figure 5-A and Figure 5-B respectively. Figure 5-C shows the candidate traceability links in an ordered list. Ranking from the top are the Java methods that have the highest textual similarity with the to-be-traced requirement. All variants of our tool compute the textual similarity by adopting the vector space model with TF-IDF weighting as its performance is comparable to other IR-based requirements tracing methods [27, 52].

Our tool structures the operation process so that the candidate links must be vetted first before the actual requirements change can be made. To do so, our tool disables the editing inside the code editor (Figure 5-D) until the participant submits a non-empty set of true links (Figure 5-E). Once the links are vetted, an ordered list of true links is shown (Figure 5-F) and the code editor becomes editable (Figure 5-G). The code editor stays editable even though the participant may modify her collection of true links (Figure 5-H), after which the ordered candidate links with her most recent vetting decisions are shown (Figure 5-C). While the

Table 1: Description of the to-be-traced requirement and the three tasks.

Req. / Task	Description
Requirement	UC11 Document office visit ... [S2] The HCP documents the following information related to an office visit: Prescribed Medications (NDC, see Data Format 6.6) Lab procedures that are ordered (LOINC code, see Data Format 6.11) ... [S4] The HCP has selected a medication prescribed from a pull down list. ...
V&V Task	Find all and only methods implementing UC11.
Change Task	Change: [S4] The HCP has selected a medication prescribed from a pull down list. To: [S4] The HCP has selected a medication prescribed from a pull down list. The medications desired to be prescribed is checked for interactions between other drugs currently taken by the patient.
Reuse Task	Reuse iTrust’s UC11 and its implementation to develop the following features in the SDMS system: Before a student graduates, the system must check whether this student meets following criteria: 1. A student must take certain courses (Data Format SDMS 3.5.4) and earn certain credits; 2. An engineering undergraduate student must take 5 co-ops (Data Format SDMS 3.5.7); 3. A graduate student must have a thesis, which requires a passed oral thesis defense. ...

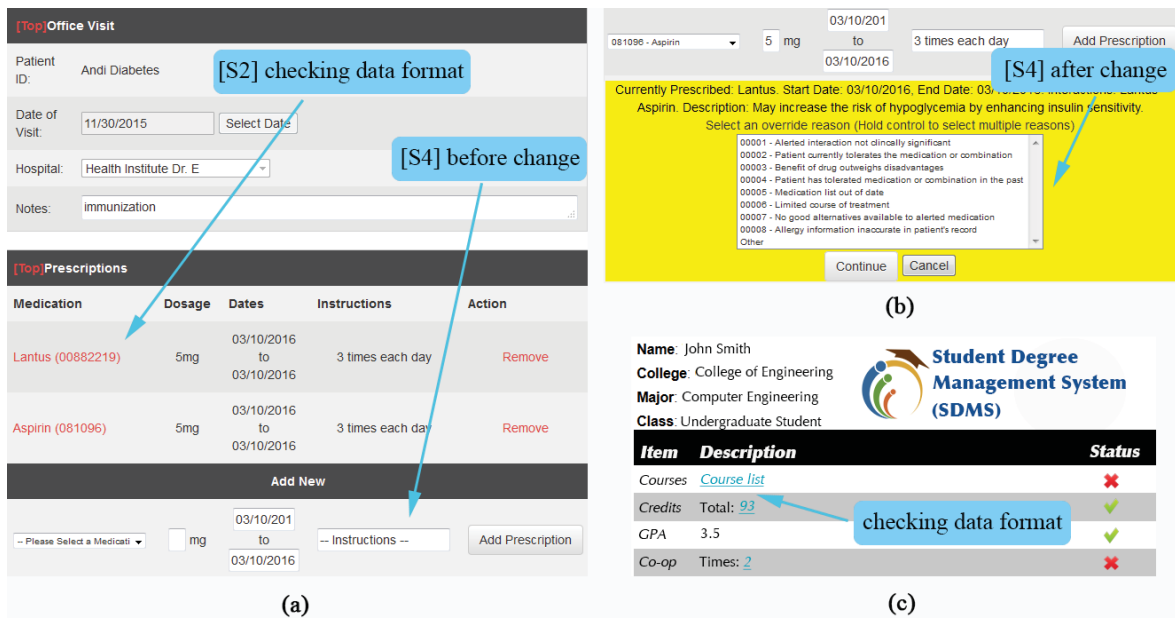
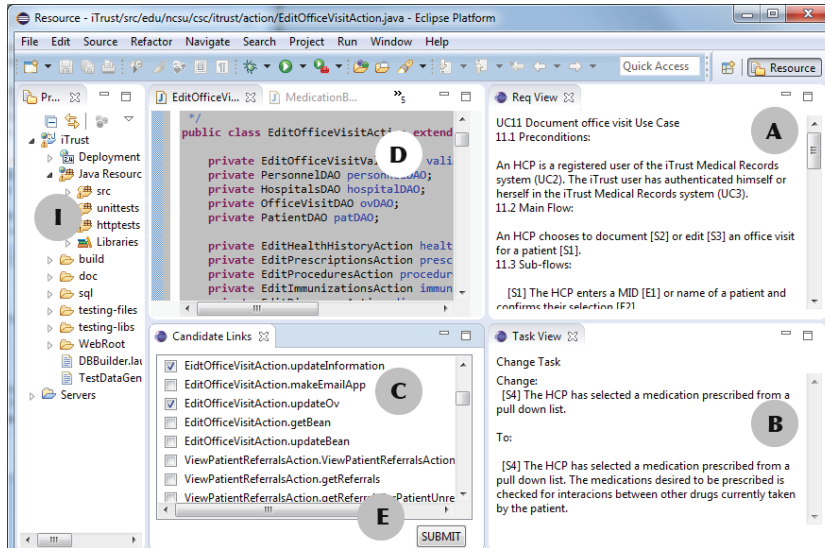


Figure 4: (a) UC11 in iTrust; (b) Changing UC11 in iTrust; (c) Reusing iTrust’s UC11 in SDMS.

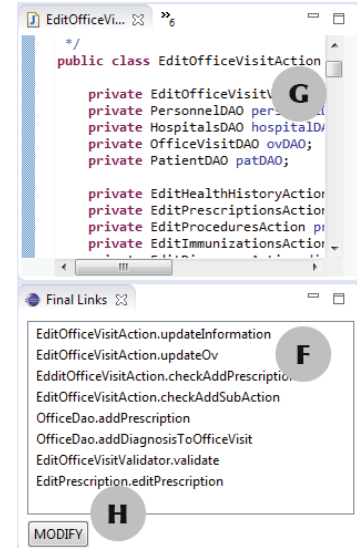
code can be navigated within the package explorer (Figure 5-I), double clicking a trace link in Figure 5-C or Figure 5-F triggers the location and the display of the link content (i.e., method body) in the code editor.

For the reuse variant of our tool shown in Figure 6, the vetting-enables-code-editing process is also enforced. Compared with Figure 5, two main differences are highlighted in Figure 6: Package explorer shows both the original project and the new project to the left of the tool (Figure 6a), and to-be-traced requirement and reuse-task description are displayed side-by-side to the right of the tool (Figure 6b). The middle part of the requirements reuse tool shares the same design as Figure 5: On the top is the one and the only one code editor in which multiple source code files can be opened and on the bottom is the switch between the two lists of the candidate links and the selected ones.

We recruited 90 undergraduate students from our university to participate in the controlled experiment. These students majored in Computer Science, Computer Engineering, or Electrical Engineering, and were recruited from a junior-level software engineering class where traceability and Java were taught. We sent e-mail invitations to all the 103 students near the end of the course. Our voluntary participants included 77 males and 13 females, had ages ranging from 19 to 28, and gained at least one 13-week semester of industrial co-op experience due to their degree programs’ requirements. None of the participants used any automated tracing tool or knew iTrust before the experiment, but all of them reported to be familiar with the healthcare domain. Because software engineering experience does not affect analysts’ tracing performance [11], we randomly assigned the 90 students into the 3 groups. While the participants in G0 were asked to

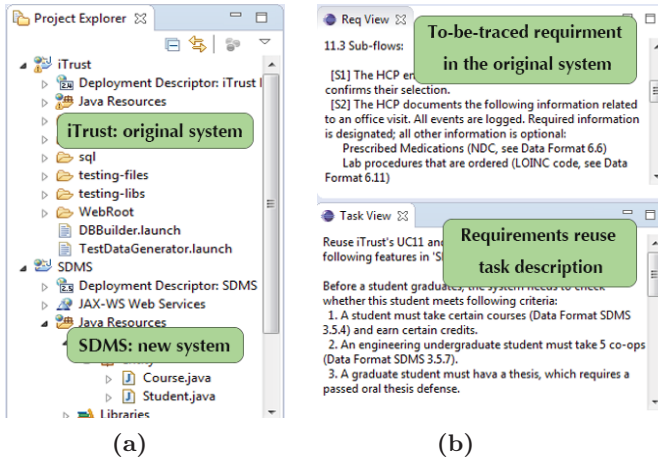


(a) Judging candidate links with explicitly defined software engineering task



(b) Using final links to solve the task

Figure 5: Screenshots of the prototype tool illustrated by the requirements change task.



(a)

(b)

Figure 6: Requirements reuse tool screenshots.

judge and finalize their true links in a V&V context, **G1** and **G2** participants were asked to vet and use the links to develop their solutions in Eclipse to fulfill the given task.

The experiment was performed in a research lab where the participants worked independently. All the participants were instructed to use only the provided variant of our tool and not to use the Internet or any other resources during the experiment. Each participant was given 30 minutes to complete the assigned task. A researcher was present in all the experiment sessions to run the tool tutorial in the beginning, to time the session, to answer questions, to take notes, and in the end of each session, to ensure every participant's interactions with the tool, along with their final links and task solutions, were properly logged.

4. RESULTS AND ANALYSIS

4.1 Testing Hypothesis H

The hypothesis **H** (cf. Section 3.1) predicts a better link-vetting performance for those analysts who know directly how their final links will be used (**G1** and **G2** in our experi-

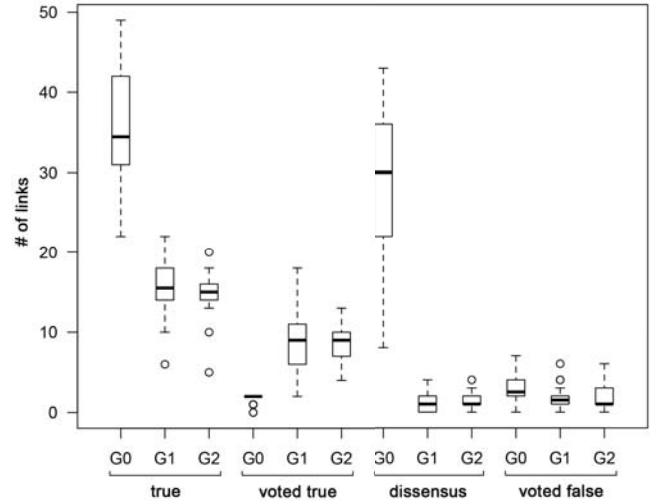


Figure 7: Number of vetted links. Number of correct links of UC11 is 20 according to the answer set defined and released by the iTrust project team.

ment) than those who know the use in a rather general way (**G0**). Figure 7 groups analyst-vetted trace links into different categories. The three leftmost box-plots of Figure 7 show the number of true links submitted by **G0**, **G1**, and **G2** participants.

Following the contemporary literature [8, 11, 26, 31, 32], we use such IR metrics as recall, precision, and F_2 to assess the analyst-vetted true links. Table 2 shows the results. Descriptive statistics are given in terms of (mean \pm standard deviation). Inferential statistics are performed via the non-parametric Wilcoxon paired difference test ($\alpha=0.05$). Table 2 provides evidence that the participants of **G2** chose more accurate traces than **G1**, whose links in turn were more accurate than **G0**'s. This indicates that to reuse, analysts need a deeper understanding than when changing an existing piece of code. Both **G1** and **G2** tasks require deeper

Table 2: Statistical results of H testing.

	G0	G1	G2
Recall	0.56±0.17	0.62±0.10	0.72±0.09
p	–	0.11	0.09
Precision	0.57±0.21	0.71±0.10	0.57±0.10
p	–	0.10	0.13
F_2	0.55±0.18	0.62±0.10	0.69±0.09
p	–	0.09	0.08

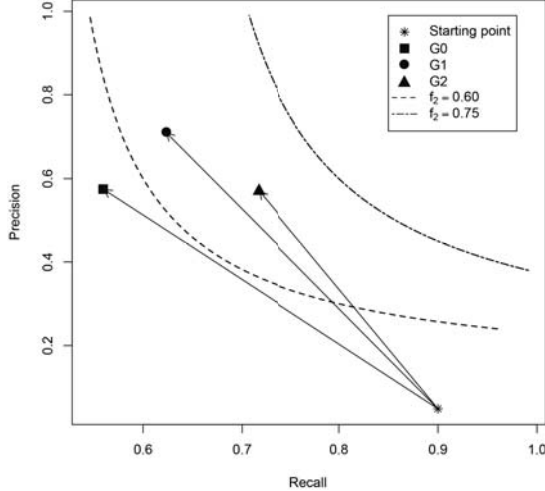


Figure 8: Average case link-vetting performance.

levels of understanding than doing the generic V&V. Nevertheless, none of the vetting performance improvements is statistically significant in that all the p values reported in Table 2 are greater than 0.05. Our results fail to reject the null hypothesis, i.e., **H** is *not* supported.

4.2 Intra-Group Agreement

To gain further insights into the link-vetting behavior, we plot the average case performance in Figure 8. All our participants started at the same point on the recall-precision space, because the candidate trace links were retrieved and ranked by the same IR algorithm. Figure 8 shows that, during vetting, our participants threw out both correct and incorrect links, enhancing precision while hurting recall⁸. The dotted lines of Figure 8 depict the F_2 -defined boundaries of what Cuddeback *et al.* [8] found to be a “hot spot” of analysts’ final links (cf. Figure 2). The prediction is amazingly good: While **G1** and **G2** finished inside the “hot spot”, **G0**’s finish was not far off at all.

We argue this predictable regularity is limited because recall and precision are set-based measures (cf. Figure 1). For example, suppose l_1 and l_2 are correct and the only correct traces of a given requirement. Let two analysts independently evaluate l_1 and l_2 . If one person marks l_1 as a true link and l_2 as a false link, and the other person marks l_1 as a false link and l_2 as a true link, then both analysts achieve the same recall (50%) and the same precision (100%). Plotted in the recall-precision space like Figure 8, the two analysts’ final traces will arrive at the same spot. However, they are in stark disagreement with each other.

To better understand our participants’ final traces, we use Fleiss’ κ [18] to directly measure the inter-rater agreement within the same group. The intra-group κ value is

⁸Our prototype tool does not support addition of the trace links that are not retrieved.

0.137, 0.437, and 0.445 for **G0**, **G1**, and **G2**, respectively. Although no generally agreed-upon measure of significance exists, a Fleiss’ κ value greater than 0.4 indicates a “good” agreement among raters [25].

The κ analysis thus reveals that knowing how the final traces are used leads **G1** and **G2** to greater intra-group consensus on what count as true links. But do different link-usage tasks lead to the same true-link consensus? The answer, as we show next, is “No”, which gives rise to the “gray links” of our interest: trace links that one group judges to be correct for their task but the other group does not.

4.3 Inter-Group Agreement and Gray Links

To determine what the true links are for the group as a whole, we adapt the two-proportion z test [58]. For a group of 30 participants rating on a link l , for example, the unanimous, 30-out-of-30 “true link” votes serve as our anchor proportion. We then test the other proportion by systematically decreasing the number of the “true link” votes: 29, 28, 27, etc. Each time this proportion changes, we run a z test to assess its difference from the anchor proportion. If the difference is statistically significant ($\alpha=0.05$ in our analyses), then a threshold is identified to signal l is no longer a group-wide “true link”. We call the true links surviving our z test *voted* true links, which are to be distinguished from the true links judged on an individual basis⁹. We apply the same z test procedure to determine the voted false links. For a given group, the traces that are voted neither as true links nor as false links are called “dissensus links”, implying the group-wide difference of opinions. Figure 7 compares the number of trace links voted by different groups.

Table 3 shows the pairwise comparisons of our three task groups. In each of the three comparisons, the top-left cell shows the proportion of the true links voted by both groups, e.g., only 2 of the 84 links (2.38%) are true links shared between **G0** and **G2** (“white links”). Similarly, 3 links (3.57%) appearing in the bottom-right cell of Table 3b are false links voted by these two groups (“black links”). The two comparisons involved **G0** result in very small proportions of white links and black links, mainly due to the low intra-group agreement of **G0** ($\kappa=0.137$). Between **G1** and **G2**, 30.23% of the links are white links meaning that both groups voted them as true links with respect to their own tasks. These groups also share 48.84% voted false links, indicating that both groups agreed that about half of the links were not relevant to either of their tasks.

The grayed cells of Table 3c represent what we call *gray links*. For **G1**, the requirements-change-task-induced gray links include 4 traces (9.30%) that **G2** could not agree on (dissensus) and 1 false trace (2.33%) voted by **G2** as a group. Similarly, the gray links of the requirements reuse task consist of the 4 (9.30%) **G1**-dissensus links. Altogether, about 20% of all the links rated by the two direct-link-use groups receive favorable “Yes, this is a link” votes from one group but not from the other.

We emphasize that the identification of gray links is very dependent on the condition that there is a reasonably large proportion of white and black links. Failing to satisfy this condition pushes many links into the hard-to-decide, “gray” set. If every link is gray, no one is. Therefore, it is not sensible to define gray links in Table 3a and Table 3b.

⁹The individually-judged true links are simply called “true links” in this paper; see Section 3.1 for definition.

Table 3: Pairwise comparison of group-wide trace links (N is the number of links shared by the two groups).

(a) G0 versus G1 ($N=82$)				(b) G0 versus G2 ($N=84$)				(c) G1 versus G2 ($N=43$)			
G0 \ G1	Voted True	Dissensus	Voted False	G2 \ G0	Voted True	Dissensus	Voted False	G2 \ G1	Voted True	Dissensus	Voted False
Voted True	1.22%	1.22%	0%	Voted True	2.38%	0%	0%	Voted True	30.23%	9.30%	2.33%
Dissensus	13.41%	2.44%	70.73%	Dissensus	10.71%	3.57%	71.43%	Dissensus	9.30%	0%	0%
Voted False	7.32%	1.22%	2.44%	Voted False	7.14%	1.19%	3.57%	Voted False	0%	0%	48.84%

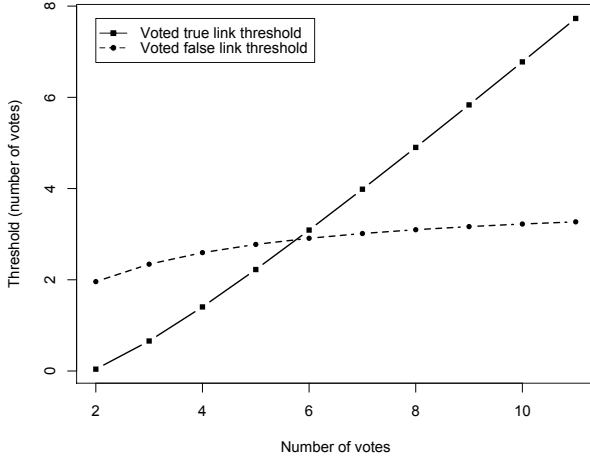


Figure 9: The z test requires at least 6 votes.

Our identification mechanism based on the z test should be treated as *a* method, not *the* method. A limitation of this method is that it requires a sufficient number of votes to sensibly distinguish the group-wide true links from false ones. Figure 9 plots two thresholds. A trace link must receive more “Yes” votes than the threshold defined by the solid line to be voted true. The opposite holds for the dotted-line-threshold of the voted false links. Before the two thresholds meet in Figure 9, a link can be voted to be both true and false—a clear contradiction. The lower bound for our gray-link identification method, therefore, is 6 votes. The method, meanwhile, has a practical upper bound, too. Figure 10 presents our analysis. As the total number of votes increases to 40, over 90% of the group opinions must be “Yes” for a link to be voted true. This threshold is too aggressive to be achieved in practice. Based on the above analyses, our z test is expected to sensibly detect gray links when the number of votes is between 6 and 40. The 30 participants (votes) in each of our three task groups fall into the desired [6, 40] range.

5. USES OF THE GRAY LINKS

Our results presented so far concern only the existence of gray links. This section analyzes how the trace links were actually used in fulfilling the tasks. Figure 11 shows the task completion time of the three task groups. For **G1** and **G2**, we also distinguish the time used by the participants who submitted correct and incorrect solutions in Figure 11. The correctness of solutions was judged by the research team according to the task descriptions given in Table 1. **G0** used less time than **G1**, which used less time than **G2**. Partici-

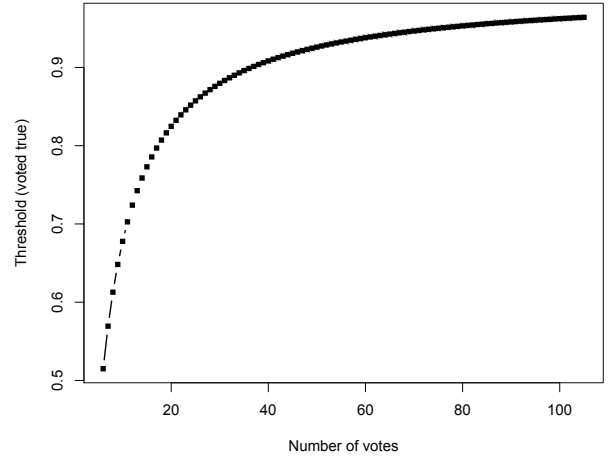


Figure 10: Practical upper bound is around 40 votes.

pants who were successful in developing their solutions spent less time than those whose task solutions were not correct.

We observed that the development of correct solutions was affected by the 30-minute time limit, as well as other factors, such as the participants’ (Java) programming skills and even their familiarity with Eclipse. As shown in Figure 11, these factors seem to negatively impact the reuse task (22/30=73.3% unsuccessful rate) more than the change task (17/30=56.7% unsuccessful rate).

The main factor in which we are interested is the actual use of the gray links, together with other types of trace links analyzed in the previous section. A trace link is *edited* means that the participant modified the Java method in the experiment. Although editing is only one kind of traceability use, we believe it is a crucial kind due to the concrete expected outcome of the change and reuse tasks.

Table 4 provides an overview of the link editing results. For both tasks, the white links received the most proportions of edits, regardless of the solution being correct or incorrect, which shows that a certain set of links is core to the given requirement. The gray links identified in our analysis also seem to be directly contributing to the task solutions. For the correct solutions, 75% and 60% of task-dependent gray links were edited. Moreover, these correct solutions did not touch the gray links belonging to the other task.

The black links were not completely unused, which is not surprising as black links still received some individual-level “Yes” votes. Interestingly, the combination of white, gray, and black links did not guarantee that a correct solution in the given time could be developed for either task. During UC11 reuse, for example, 11 trace links were edited but

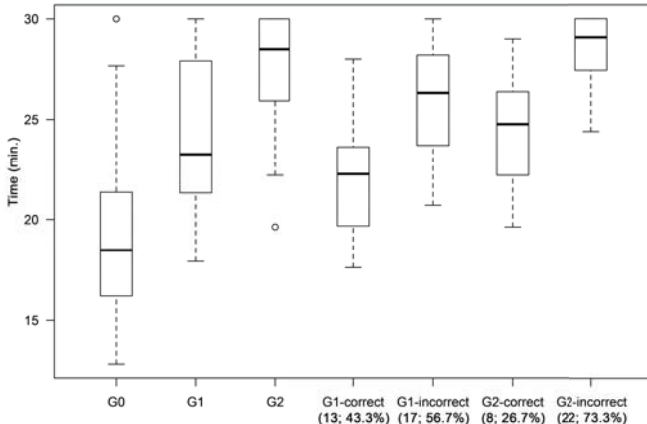


Figure 11: Task completion time.

Table 4: Number of edited trace links.

	Change task solutions		Reuse task solutions	
	Correct	Incorrect	Correct	Incorrect
White (13)	12 (92%)	9 (69%)	10 (77%)	5 (38%)
Gray of G1 (4)	3 (75%)	2 (50%)	0 (0%)	1 (25%)
Gray of G2 (5)	0 (0%)	0 (0%)	3 (60%)	1 (20%)
Black (21)	1 (5%)	4 (19%)	2 (10%)	6 (29%)
Not judged in G1 (7)	6 (86%)	4 (57%)	–	–
Not judged in G2 (11)	–	–	7 (64%)	5 (45%)

never judged to be correct by any participant from **G2**. The main reason, according to our observations, was that none of the participants went back to revise their judgment once they submitted their vetting results. Those 11 links were uncovered and edited during the actual requirements reuse. Finally, no participant ever edited any link not retrieved by our TF-IDF algorithm, which implies that if the participants want to go back and revise their link-vetting results, they would find their edited links inside the candidate list. This result also shows that even though the TF-IDF algorithm did not retrieve all the correct links defined in the answer set, the 90%-recall performance (cf. Figure 8) was practically adequate for solving both **G1** and **G2** tasks.

In summary, the gray links identified in our earlier analysis were both viewed and edited to a great extent during the performance of the specific tasks. Not only were the gray links task-dependent, so were their inter-relations. We observed that method calls were particularly helpful for supporting the requirements change task, whereas method encapsulations were useful for supporting the requirements reuse task. The call relations have recently been leveraged to automatically uncover a more complete set of requirements-to-code traces [19]. We believe the encapsulation relations can facilitate further automation, especially for those tasks requiring a relatively higher-level grasp of the software structure, such as reuse, reserve engineering, and refactoring.

6. DISCUSSIONS

6.1 Threats to Validity

As is the case for most controlled experiments, our investigation into trace-use-task-induced gray links is performed in a restricted and synthetic context. We discuss some of the most important factors that must be considered when interpreting the results.

Study of only a single iTrust requirement. We chose UC11 because of its representativeness in the evolution of iTrust. Based on the 15 iTrust releases from 2007 to 2015 [59], the number of changes of all the 34 UCs is 7.62 ± 3.67 whereas that of UC11 is 8. The number of changed trace links (Java methods) of all the UCs is 9.24 ± 3.24 and that of UC11 is 8.62 ± 2.13 . In addition to change frequency and size, the representativeness of UC11 is reflected in the number of trace links defined in the answer set: 17.82 ± 10.69 for all the UCs and 20 for UC11. However, UC11 may not be representative in many other respects: change complexity, priority, error-proneness, etc. Examining only one requirement of only one software system limits the generalizability of our study.

Selection of the three tasks. Our choice of the three tasks was motivated by the directness of traceability use. We considered V&V a *de facto* way of “vetting-links-for-the-sake-of-vetting-links” as appeared in many previous studies [8, 31, 32, 48, 60]. Recent work by Mäder and Egyed [34] made more direct use of requirements traceability to support software change tasks. Extending [34], we examined the software change not only within the original system but toward the reuse in a different system. While our work indicates that different levels of understanding are needed for different tasks, our findings may not generalize to other tasks. Investigating more task types like refactoring [44], creativity [3], collaboration [4], and maybe even a participant-group without any task will gain insights into how related tasks affect the gray-link judgment.

Participants having no project knowledge. Because the traceability answer set is accessible from iTrust Web site, we required that the participants had no project knowledge before the experiment. For the same reason, we restricted each participant to use only our prototype tool during the experiment. The settings are not representative of documenting traceability information in industry, especially in safety-critical projects [35]. Our hypothesis is that gray links would still be present (e.g., when a regulatory requirement is assured or updated), but the identification of the gray links might require different tools and occur at different time scales. Even for controlled settings like a lab experiment, the threat can be addressed by having the participants acquiring project familiarity first and then documenting the traces they consider important for the task.

Impact of the 30-minute time limit. The time constraint we imposed on each task, though informed by Mäder and Egyed’s study [34], was artificial. In [34], a 30-minute limit was set for every software change task and most of the participants completed a task within 10-15 minutes. However, the participants of [34] needed to only sketch the task solutions on paper, whereas our **G1** and **G2** participants tried to make their solutions work in Eclipse. In addition, neither V&V nor reuse tasks were investigated in [34]. According to Figure 11, V&V took the least time while reuse took the most. The 30-minute limit may have led us to observe participants’ *initial* trace-link vetting and using behaviors. The time limit is also a potential confounding variable for the observed higher number of trace links submitted by **G0** shown in Figure 7. Future (quasi)replications should consider relaxing or completely removing the task time-to-completion constraint.

6.2 Implications

Our study, to the best of our knowledge, is the first to bring concrete evidence on the importance of the task at end when it comes to traceability recovery. We discuss how our findings of trace-use-task-induced gray links influence the contemporary literature.

Evaluating trace retrieval methods. Standard IR metrics like precision and recall are used to evaluate IR at the *output* level [55]. As shown in Figure 1, the output-level evaluation requires a predefined answer set. In automated traceability, an authoritative and updated answer set rarely exists [59]. Our discovery of gray links offers a new, flexible, and pragmatic way to evaluate IR-based trace retrieval methods. Figure 12 illustrates the idea. Rather than having a unique answer set, the trace links can be organized into white links that may be central to understanding the implementation of the given requirement, and gray links that are valuable for solving specific tasks (or sets of related tasks) involving the requirement. In this way, the evaluation can be conducted toward the *use* and *user* level [55].

From vetting the links to using the links. The work on analysts’ link-vetting behavior [8, 11, 26, 31, 32] led us to consider explicitly the human-side of tracing. In these studies, however, the links were never directly used beyond the analysts’ final approval. We suggest a direct trace use process to further tighten the analyst–tool interaction in automated traceability. As our results show, certain trace links were edited during use but were not re-vetted to reflect analysts’ judgment. Our prototype tool, in this regard, needs improvement. For example, by actively monitoring development environments [2], more accurate and more complete traces could be built on-the-fly instead of after-the-fact through vetting. Although students participating in prior traceability research are common [8, 11, 16, 31, 32, 48, 60], we realize that they fit more into the profile of *low-end* users, according to Ramesh and Jarke’s classification [54]. While low-end users need training and experience to become high-end users, we stress that the students in our study are *users*, instead of just *vetters*, of traceability.

Treating trace use as a first-class citizen. The traceability life cycle is typically described as creating, maintaining, and using trace links while planning and managing a traceability strategy [6, 23]. We believe that listing “using” after “creating and maintaining” is accidental. In fact, having *purposed* traceability is identified as the number one grand challenge facing the community [6, 22]. Our work on using the traceability to solve different tasks on the same requirement illuminates that purposed traceability must treat usage tasks and scenarios early in the traceability life cycle. In our opinion, if the trace creator and maintainer are *not* the same as the trace users, then traceability is likely to stay as one of the most elusive qualities of the software development in practice [6]. Having a constant or even an enforced trace-use mentality will not only help create trace links fitting better a specific purpose, but also challenge some advocated best practices. Take predefining a traceability information model (TIM) [53] for example, our results show that, for different trace-use tasks, specifying a single TIM at a fixed granularity level (e.g., “UC-to-method”) may be counterproductive. Instead, a TIM of “subflow-of-UC-

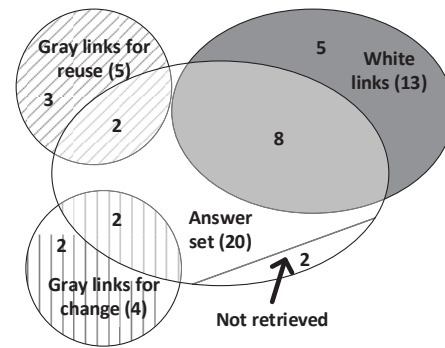


Figure 12: Evaluating IR-based trace retrieval at the trace *use* and *user* level.

to-method” can be more suitable for requirements change whereas a TIM of “UC-to-class” may be more appropriate for requirements reuse.

7. CONCLUSIONS

“Not all trace links are equally important” is the vision of value-based software traceability [15]. To realize this vision, we investigated in this paper the impact of different software engineering tasks on tracing the same requirement. Our results show that analysts who know how their final links will be used tend to reach consensus on what count as true links. However, the consensus is *not* shared between groups with different tasks, giving rise to gray links dependent on tasks. We further contribute a mechanism to identify gray links and discuss how to exploit them to better support specific requirements change and reuse tasks.

Substantial limitations exist in our study, which we plan to address as part of the future work. Replicating the experiment without task-completion time constraint and carrying out more in-depth empirical studies—with other iTrust requirements than UC11, with other systems than iTrust, and with other traceability users than students having no project knowledge—are in order. Our future work also includes improving the gray-link identification method *a priori* or by leveraging evidence and “votes” from multiple sources, such as version control logs [44], potentially monitored development environments [2], or even outdated trace links [59]. Finally, we are interested in extending, expanding, and enhancing the gray links toward developers’ daily tasks (e.g., refactoring and code review), which are not only commonly assigned to them but are also integral to their own perceptions of productivity [42].

8. ACKNOWLEDGMENTS

We thank all the participants of our study. We also thank the anonymous reviewers for the valuable and constructive feedback. The work is funded in part by the U.S. NSF (National Science Foundation) Grants CCF-1350487 and CCF-1623089.

9. REFERENCES

- [1] N. Ali, Y.-G. Guéhéneuc, and G. Antoniol. Trustrace: mining software repositories to improve the accuracy of requirement traceability links. *IEEE Transactions on Software Engineering*, 39(5):725–741, May 2013.

- [2] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor. Software traceability with topic modeling. In *ICSE*, pages 95–104, Cape Town, South Africa, May 2010.
- [3] T. Bhowmik, N. Niu, A. Mahmoud, and J. Savolainen. Automated support for combinational creativity in requirements engineering. In *RE*, pages 243–252, Karlskrona, Sweden, August 2014.
- [4] T. Bhowmik, N. Niu, W. Wang, J.-R. C. Cheng, L. Li, and X. Cao. Optimal group size for software change tasks: a social information foraging perspective. *IEEE Transactions on Cybernetics*, 46(8):1784–1795, August 2016.
- [5] M. Borg, P. Runeson, and A. Ardö. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering*, 19(6):1565–1616, December 2014.
- [6] J. Cleland-Huang, O. Gotel, J. H. Hayes, P. Mäder, and A. Zisman. Software traceability: trends and future directions. In *FOSE*, pages 55–69, Hyderabad, India, May-June 2014.
- [7] J. Cleland-Huang, G. Zement, and W. Lukasik. A heterogeneous solution for improving the return on investment of requirements traceability. In *RE*, pages 230–239, Kyoto, Japan, September 2004.
- [8] D. Cuddeback, A. Dekhtyar, and J. H. Hayes. Automated requirements traceability: the study of human analysts. In *RE*, pages 231–240, Sydney, Australia, September-October 2010.
- [9] D. Cuddeback, A. Dekhtyar, J. H. Hayes, J. Holden, and W.-K. Kong. Towards overcoming human analyst fallibility in the requirements tracing process. In *ICSE*, pages 860–863, Honolulu, HI, USA, May 2011.
- [10] T. Dasgupta, M. Grechanik, E. Moritz, B. Dit, and D. Poshyvanik. Enhancing software traceability by automatically expanding corpora with relevant documentation. In *ICSM*, pages 320–329, Eindhoven, The Netherlands, September 2013.
- [11] 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*, pages 111–120, Trento, Italy, August-September 2011.
- [12] A. Dekhtyar, J. H. Hayes, and J. Larsen. Make the most of your time: how should the analyst work with automated traceability tools? In *PROMISE*, Minneapolis, MN, USA, May 2007.
- [13] C. Duan and J. Cleland-Huang. Clustering support for automated tracing. In *ASE*, pages 244–253, Atlanta, GA, USA, November 2007.
- [14] A. Egyed. Resolving uncertainties during trace analysis. In *FSE*, pages 3–12, Newport Beach, CA, USA, October-November 2004.
- [15] A. Egyed, S. Biffi, M. Heindl, and P. Grünbacher. A value-based approach for understanding cost-benefit trade-offs during automated software traceability. In *TEFSE*, pages 2–7, Long Beach, CA, USA, November 2005.
- [16] A. Egyed, F. Graf, and P. Grünbacher. Effort and quality of recovering requirements-to-code traces: two exploratory experiments. In *RE*, pages 211–230, Sydney, Australia, September-October 2010.
- [17] A. Egyed, P. Grünbacher, M. Heindl, and S. Biffi. Value-based requirements traceability: lessons learned. In *RE*, pages 115–118, New Delhi, India, October 2007.
- [18] J. L. Fleiss and J. Cohen. The equivalence of weighted kappa and the intraclass correlation coefficient as measures of reliability. *Educational and Psychological Measurement*, 33(3):613–619, October 1973.
- [19] A. Ghabi and A. Egyed. Code patterns for automatically validating requirements-to-code traces. In *ASE*, pages 200–209, Essen, Germany, September 2012.
- [20] A. Ghabi and A. Egyed. Exploiting traceability uncertainty between architectural models and code. In *WICSA/ECSA*, pages 171–180, Helsinki, Finland, August 2012.
- [21] A. Ghabi and A. Egyed. Exploiting traceability uncertainty among artifacts and code. *Journal of Systems and Software*, 108:178–192, October 2015.
- [22] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, and J. I. Maletic. The grand challenge of traceability (v1.0). In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 343–409. Springer, 2012.
- [23] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, J. I. Maletic, and P. Mäder. Traceability fundamentals. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 3–22. Springer, 2012.
- [24] O. Gotel and A. Finkelstein. An analysis of the requirements traceability problem. In *ICRE*, pages 94–101, Colorado Springs, CO, USA, April 1994.
- [25] K. L. Gwet. *Handbook of Inter-Rater Reliability*. Advanced Analytics, 2014.
- [26] J. H. Hayes and A. Dekhtyar. Humans in the traceability loop: can't live with 'em, can't live without 'em. In *TEFSE*, pages 20–23, Long Beach, CA, USA, November 2005.
- [27] 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*, 32(1):4–19, January 2006.
- [28] A. Jain and B. Boehm. Developing a theory of value-based software engineering. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–5, July 2005.
- [29] X. Jin, C. Khatwani, N. Niu, M. Wagner, and J. Savolainen. Pragmatic software reuse in bioinformatics: how can social network information help? In *ICSR*, pages 247–264, Limassol, Cyprus, June 2016.
- [30] D. Kelly and X. Fu. Elicitation of term relevance feedback: an investigation of term source and context. In *SIGIR*, pages 453–460, Seattle, WA, USA, August 2006.
- [31] W.-K. Kong, J. H. Hayes, A. Dekhtyar, and O. Dekhtyar. Process improvement for traceability: a study of human fallibility. In *RE*, pages 31–40, Chicago, IL, USA, September 2012.
- [32] 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*,

- pages 32–39, Honolulu, HI, USA, May 2011.
- [33] P. Lago, H. Muccini, and H. van Vliet. A scoped approach to traceability management. *Journal of Systems and Software*, 82(1):168–182, January 2009.
- [34] P. Mäder and A. Egyed. Do developers benefit from requirements traceability when evolving and maintaining a software system? *Empirical Software Engineering*, 20(2):413–441, April 2015.
- [35] P. Mäder, P. L. Jones, Y. Zhang, and J. Cleland-Huang. Strategic traceability for safety-critical projects. *IEEE Software*, 30(3):58–66, May/June 2013.
- [36] A. Mahmoud and N. Niu. On the role of semantics in automated requirements tracing. *Requirements Engineering*, 20(3):281–300, September 2015.
- [37] A. Mahmoud, N. Niu, and S. Xu. A semantic relatedness approach for traceability link recovery. In *ICPC*, pages 183–192, Passau, Germany, June 2012.
- [38] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
- [39] A. Marcus and J. I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *ICSE*, pages 125–137, Portland, OR, USA, May 2003.
- [40] C. McMillan, D. Poshyvanyk, and M. Reville. Combining textual and structural analysis of software artifacts for traceability link recovery. In *TEFSE*, pages 41–48, Vancouver, Canada, May 2009.
- [41] A. Meneely, B. Smith, and L. Williams. iTrust electronic health care system case study. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 425–438. Springer, 2012.
- [42] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann. Software developers’ perceptions of productivity. In *FSE*, pages 19–29, Hong Kong, China, November 2014.
- [43] L. Moreno, G. Bavota, S. Haiduc, M. Di Penta, R. Oliveto, B. Russo, and A. Marcus. Query-based configuration of text retrieval solutions for software engineering tasks. In *FSE*, pages 567–578, Bergamo, Italy, August-September 2015.
- [44] N. Niu, T. Bhowmik, H. Liu, and Z. Niu. Traceability-enabled refactoring for managing just-in-time requirements. In *RE*, pages 133–142, Karlskrona, Sweden, August 2014.
- [45] N. Niu, X. Jin, Z. Niu, J.-R. C. Cheng, L. Li, and M. Y. Kataev. A clustering-based approach to enriching code foraging environment. *IEEE Transactions on Cybernetics*, (to appear).
- [46] N. Niu and A. Mahmoud. Enhancing candidate link generation for requirements tracing: the cluster hypothesis revisited. In *RE*, pages 81–90, Chicago, IL, USA, September 2012.
- [47] N. Niu, A. Mahmoud, and G. Bradshaw. Information foraging as a foundation for code navigation. In *ICSE*, pages 816–819, Honolulu, HI, USA, May 2011.
- [48] N. Niu, A. Mahmoud, Z. Chen, and G. Bradshaw. Departures from optimality: understanding human analyst’s information foraging in assisted requirements tracing. In *ICSE*, pages 572–581, San Francisco, CA, USA, May 2013.
- [49] N. Niu, A. Mahmoud, and X. Yang. Faceted navigation for software exploration. In *ICPC*, pages 193–196, Kingston, Canada, June 2011.
- [50] N. Niu, S. Reddivari, and Z. Chen. Keeping requirements on track via visual analytics. In *RE*, pages 205–214, Rio de Janeiro, Brazil, July 2013.
- [51] N. Niu, J. Savolainen, Z. Niu, M. Jin, and J.-R. C. Cheng. A systems approach to product line requirements reuse. *IEEE Systems Journal*, 8(3):827–836, September 2014.
- [52] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia. On the equivalence of information retrieval methods for automated traceability link recovery. In *ICPC*, pages 68–71, Braga, Portugal, June 2010.
- [53] F. A. C. Pinheiro and J. A. Goguen. An object-oriented tool for tracing requirements. *IEEE Software*, 13(2):52–64, March 1996.
- [54] B. Ramesh and M. Jarke. Toward reference models of requirements traceability. *IEEE Transactions on Software Engineering*, 27(1):58–93, January 2001.
- [55] T. Saracevic. Evaluation of evaluation in information retrieval. In *SIGIR*, pages 138–146, Seattle, WA, USA, July 1995.
- [56] A. Sardinha, Y. Yu, N. Niu, and A. Rashid. Ea-tracer: identifying traceability links between code aspects and early aspects. In *SAC*, pages 1035–1042, Trento, Italy, March 2012.
- [57] H. Sultanov, J. H. Hayes, and W.-K. Kong. Application of swarm techniques to requirements tracing. *Requirements Engineering*, 16(3):209–226, September 2011.
- [58] R. E. Walpole, R. H. Myers, S. L. Myers, and K. E. Ye. *Probability and Statistics for Engineers and Scientists*. Pearson, 2011.
- [59] W. Wang, A. Gupta, and Y. Wu. Continuously delivered? periodically updated? never changed? studying an open source project’s releases of code, requirements, and trace matrix. In *JITRE*, pages 13–16, Ottawa, Canada, August 2015.
- [60] W. Wang, N. Niu, H. Liu, and Y. Wu. Tagging in assisted tracing. In *SST*, pages 8–14, Florence, Italy, May 2015.