Development of Robust Traceability Benchmarks

Xiaofan Chen
Department of Computer Science,
University of Auckland,
Auckland, New Zealand,
xche044@aucklanduni.ac.nz

John Hosking
College of Engineering & Computer Science
Australian National University,
Canberra, Australia
john.hosking@anu.edu.au

John Grundy
Centre for Computing and Engineering Software Systems
Swinburne University of Technology,
Hawthorn, Australia
jgrundy@swin.edu.au

Robert Amor
Department of Computer Science,
University of Auckland,
Auckland, New Zealand,
trebor@cs.auckland.ac.nz

***Abstract*—Traceability benchmarks are essential for the evaluation of traceability recovery techniques. This includes the validation of an individual traceability technique itself and the objective comparison of the technique with other traceability techniques. However, it is generally acknowledged that it is a real challenge for researchers to obtain or build meaningful and robust benchmarks. This is because of the difficulty of obtaining or creating suitable benchmarks. In this paper, we describe an approach to enable researchers to establish affordable and robust benchmarks. We have designed rigorous manual identification and verification strategies to determine whether or not a link is correct. We have developed a formula to calculate the probability of errors in benchmarks. Analysis of error probability results shows that our approach can produce high quality benchmarks, and our strategies significantly reduce error probability in them.**

Keywords—traceability benchmark; benchmark development

I. INTRODUCTION

It is well recognized that rigorous traceability technology plays a critical role in the software development process [1, 2, 8, 19]. Such support helps developers to assess the completeness of an implementation against all stated requirements, to identify reusable software components, to support impact analysis while changes occur, and to comprehend and maintain systems [1, 2, 19]. In the last decade, researchers have put great effort into inventing new traceability recovery techniques [1, 11, 13, 14, 20, 23], either automatic or semi-automatic, to retrieve traceability links in a software system with as many correct links and as few incorrect links as possible. An outstanding research question is how to examine the performance of a recovery technique, validate the technique, and compare its performance with other recovery techniques. An essential element for the traceability recovery technique evaluation process is the use of effective traceability benchmarks [7, 9, 21]. A traceability benchmark serves as a basis for evaluation of one technique and the comparison of multiple techniques [7]. Without such a benchmark, whether a link generated by the recovery technique is correct or not remains uncertain, and whether any correct links are missed is unclear. Without such data, the performance of a recovery technique cannot be measured or compared with the performance of others. In addition, improvements to the recovery technique cannot be determined or quantified. However, it is a major challenge for researchers to obtain or establish meaningful and robust benchmarks to evaluate recovery approaches, due to the difficulty of obtaining or creating such benchmarks [8].

Traceability benchmarks can be acquired or developed in three ways. First, a benchmark for a project can be built by project developers during the software development process. This type of benchmark is the most attractive due to its apparent reliability and accuracy. Unfortunately, in practice traceability within the software development process has seldom been employed in most organizations. This is due to its high cost, complexity, time-consuming nature, and error-proneness [10, 16, 17]. This results in there being almost no publicly available projects containing very robust benchmarks [5]. Second, we can evaluate a new recovery technique with benchmarks developed by other researchers and/or applied by them to other software projects. However, a major problem with this approach is that different recovery techniques often address different issues and are applied to different artifacts. This results in limited ability to adapt or use benchmarks from the work of others [9]. For example, a benchmark that was built to include links between requirements and design documents, while very useful for testing recovery techniques between these types of artifacts, is not at all suitable to evaluate a recovery technique that captures links between source code and documentation or between source code and unit tests. Third, we can establish our own benchmarks to meet our specific recovery technique evaluation needs. Due to the well-known difficulty of obtaining or using the above two types of benchmarks, researchers usually create their own benchmarks to conduct evaluations of their new recovery techniques [5, 8, 9, 11]. Nevertheless, a major issue arises here: how do we actually go about building affordable and robust traceability benchmarks? Most research to date has focused on the theory, basic principles, or the establishment of all-inclusive traceability benchmarks [5, 9]. Unfortunately, there are no generally agreed or applied approaches or guidelines that have been proposed to assist researchers in manually building robust benchmarks for identifying and verifying links in a system.

In this paper, we propose a new approach and guidelines to help researchers who want to develop their own traceability benchmarks and use these to facilitate evaluation and comparison of their own and others traceability recovery techniques. The main objective of a recovery technique is to trace links between artifacts in a system. A new recovery technique is normally evaluated by comparing the set of its retrieved links with an "oracle" i.e. the known, true link set of

the system, in order to compute its precision, recall, or F-measure. The oracle link set of a system consists of all of the actual correct links for the system. Hence, the crucial part of the development of traceability benchmarks is manually finding and verifying correct links between artifacts. We have designed and tested rigorous manual identification and verification strategies to assist researchers to capture links and verify them. In our approach, every link is analyzed by at least two analysts before the determination of its status is made i.e. whether it is a correct/true or incorrect link. We propose a formula to calculate the probability of errors (e.g. 5% or 10% erroneous links) in the created oracle link set. We have employed this approach to create a very robust benchmark for JDK1.5. We employed the formula to compute the probabilities of 5% and 10% errors in our JDK1.5 oracle link set. The result shows that the actual probability of there being $\geqslant$ 5% error links in this oracle link set is very low, around 0.1%. Moreover, our error probability calculation shows that our rigorous manual identification and verification strategies can significantly reduce the error probability in the oracle link set.

This paper is organized as follows. We firstly provide a background to traceability benchmarks and the motivation for this research. We introduce our approach to establishing a traceability benchmark and a formula to calculate its error probability. Next we describe a benchmark that we developed using our new approach. We then compute the probability of errors in our created JDK1.5 oracle link set. We also analyze the cost-quality tradeoffs in establishing a robust traceability benchmark. Finally, we discuss the limitations of our approach and threats to validity.

II. Background and Motivation

Sim et al. [21] define a benchmark in software engineering as a standard test or set of tests employed to compare the performance of alternative tools or techniques. They claim that successful benchmarks must meet seven requirements: accessibility, affordability, clarity, relevance, solvability, portability, and scalability. Oppenheimer et al. [15] claim that benchmarks provide researchers with an approach to quantify design tradeoffs and a yardstick to measure and inspire progress. Because of the importance of benchmarks in the evaluation process, Cleland-Huang et al. [8] identify building traceability benchmarks as one of the key challenge areas in their Grand Challenges in Traceability.

Few papers have been published to tackle this challenge in the software traceability area. In the Grand Challenges in Traceability, a traceability benchmark is defined in terms of a traceability task, a set of data sets on which the task is to be performed, and finally a set of metrics that will be used to evaluate the task [8]. Dekhtyar et al. [9] establish the basic principle of organization of traceability benchmarks, which comprises five components: dataset, tasks, measures, answer sets, and data representation format (the latter is optional). To be successful, traceability benchmarks need to satisfy five requirements: support for traceability in multiple fields of software engineering, independence of methodology, ground truth, accuracy testing, and scalability testing [9]. According to this principle, Charrada et al. [5] propose a traceability benchmark that includes nine types of artifacts and with end-to-end traceability links. This benchmark is developed based on AquaLush, an irrigation system. A visual experimental workbench, named TraceLab, was developed for designing and executing traceability experiments to help new researchers to establish research environments or help existing researchers to perform more rigorous evaluations [7].

Although researchers have put some effort into building meaningful traceability benchmarks, there are three barriers impeding the realization of this challenge. The first barrier is the lack of publicly available projects that include traceability links. In general, open source projects exclude links between artifacts or contain only partial traceability. The benchmarks of commercial projects are confidential in most cases. Moreover, it is difficult to accomplish traceability in practice because tracing and maintaining links is arduous, time-consuming, error-prone, and costly [10, 16, 17]. These issues make it difficult to acquire publicly available benchmarks [5, 7].

The second challenge is the diversity of traceability issues tackled by traceability recovery techniques. Most recovery techniques address specific traceability problems because of the researchers' expertise or project funding [9]. For example, Antoniol et al. [1] use Probabilistic Model and Vector Space Model to recover links between code and documentation to assist maintainers. Marcus and Maletic [14] introduce Latent Semantic Indexing to improve the performance of Information Retrieval models. Hayes et al. [11] aim to improve the requirements tracing process for independent verification and validation analysts. Besides targeting different traceability issues, recovery techniques utilize different artifacts to examine and evaluate their performance. For instance, Antoniol et al. [1] and Marcus and Maletic [14] focus on tracing links between code and documentation. Hayes et al. [11] recover links between requirements and design specifications. Bacchelli et al. [4] seek to find links between code and emails. Moreover, researchers need to build tools implementing their recovery techniques in order to perform the evaluation. These tools may accept limited programming languages. For example, our own traceability recovery tool only traces links in software projects that are written in Java [6]. In addition, the language used to write comments and documents generated during the software development process affects the possibility of the adaptation or employment by other traceability techniques. For example, Documents in Albergate utilized by Antoniol et al. [1] and Marcus and Maletic [14] are written in Italian. For researchers who are not familiar with Italian, it is difficult to process these documents. These issues lead to difficulty when adapting benchmarks from others' work.

The last barrier is the difficulty of manually establishing robust traceability benchmarks [5, 11]. Recovery techniques mainly involve tracing links between artifacts in a system [9]. Hence, the most important part of a benchmark during evaluation and comparison of recovery techniques is the oracle, or true traceability link set, which is a set of correct/true links between artifacts. However, manually tracing links from one artifact to another is arduous, time-consuming, and error-prone. Due to the first two hindrances, researchers often have to develop their own benchmarks to meet their specific needs. For example, Hayes et al. [11] built a benchmark for the CM-1 data set to evaluate the tracing between requirements and design

documents. They used a group of analysts to manually verify links retrieved by RETRO [12], a special-purpose tool designed exclusively for requirements tracing. Bacchelli et al. [3, 4] establish five benchmarks (for ArgoUML, Freenet, JMeter, Mina, and OpenJPA) to target links between code and emails. They manually annotated classes mentioned in emails and verified these annotations with a group of six participants. There are three key issues that emerge while manually creating traceability benchmarks: how to find an appropriate dataset, how to manually identify correct links between artifacts, and how to verify links to be correct or incorrect. The three issues have rarely been touched on in the software traceability community. Moreover, there are currently no guidelines or approaches that have been proposed to assist researchers in developing meaningful and robust traceability benchmarks. These challenges motivated us to develop an approach to establish affordable, meaningful, and robust traceability benchmarks easily and effectively.

III. Traceability Benchmark Development

As mentioned earlier, Dekhytar et al. [9] define a traceability benchmark to consist of five main components: dataset, tasks, answer sets, measures, and (optionally) data representation format. We employ this definition to design our traceability benchmarks. In our research, we are concerned with the evaluation of traceability recovery techniques. These generally deal with the issue of the recovery of links between artifacts. Hence, we define a traceability benchmark to include tasks, dataset, oracle/true traceability link sets, and measures. We replace answer sets with oracle/true traceability link sets to satisfy our concerns. We propose five steps to establish a traceability benchmark: task identification, artifact selection, project selection, oracle/true traceability link set development, and evaluation metrics. The following sections describe these five steps in detail.

A. Task Identification

The first step for building a traceability benchmark is to address what tasks the benchmark is developed to accomplish. This depends upon what issues the recovery technique under evaluation is concentrating on. For example, some recovery techniques focus on tracing links between requirements and design documents, some aim to find links between source code and documents, and so on. The tasks must reflect these issues. For instance, if the evaluated recovery technique aims to trace links between source code and documentation, then the task is to recover links between code and documents.

B. Artifact Selection

Based on the tasks, the second step is to choose appropriate artifacts. If one of the tasks is the recovery of links between source code and documentation, then the dataset should be the collection of code files and documents that are produced during the software development process. To decide on what kinds of documents to select relies on their availability in the selected project. If the selected project only provides requirements and design documents, then we can only trace links between code and requirements and/or design documents. Furthermore, using a particular document detail level e.g. the class or method level for code, and the section or paragraph level for documents, is decided by the particular recovery technique that needs to be evaluated. If the traceability technique retrieves links between classes and sections in documents, then the artifact dataset includes classes and sections of documents.

C. Project Selection

The next step is to search for appropriate projects from which to obtain artifacts to analyze. An appropriate project possesses four properties. 1) The project must include artifacts that are chosen in step two. For example, if the artifacts selected in step two are requirements and design documents, then the selected project must contain the two types of documents. 2) The project should be of a reasonable size. The larger the size of the project is, the more resources, time, and cost are required to manually build links. If the benchmark builder has limited resources, then one of the feasible ways is to choose projects of small size or to use a part of large projects. For instance, if a project contains a large number of requirements and design documents, a feasible way is to choose a part of the requirements and their corresponding design documents. 3) The programming language used is accepted by the tool that implements the traceability technique under evaluation. Some such tools have no requirements on programming languages. Hence, the chosen projects can be written in any programming language. But others may only accept certain programming languages, such as Java, C#, or C++ etc. This requires that the chosen projects must be written in a language that can be accepted by the tool. 4) Documents and comments in the project must be written in a language that can be understood by any participant involved in benchmark development. For example, if documents and comments in a project are written in French, it is impossible for participants who cannot understand French to identify links in this project.

D. Oracle/True Link Set Development

After selecting the appropriate projects and artifacts, we can start building the oracle link set. At the start, we need to decide the source artifact and the target artifact. All links are bidirectional, so defining the source and target artifacts is principally to help users to more easily identify links. For the sake of simplicity, the source artifact should be the less complicated one, or the one that can be easily divided into sets or groups. For example, when recovering links between code and documents, source code is normally used as the source artifact. However, in [4], emails are used as the source artifact because their source code includes many versions of ArgoUML. Next, traceability rules need to be set up to facilitate the identification and verification of links. We apply the rule defined by [5] as fundamental: an element A and an element B are related if B is derived from A or if B provides additional and useful information about A. This rule can be extended to satisfy specific concerns. For example, for defining links between code and documents, this rule can be changed to: a class A and a section B in documents are related if B directly mentions A's name or identifier, or if B describes tasks that A should fulfill. Then we can start manually identifying and verifying links between selected artifacts in the selected project.

Figure 1 illustrates our strategies for oracle link set development. Our link identification and verification strategies are inspired by the works of [3] and [11], who establish their own benchmarks through manually verifying links by a group of participants. Our strategies are designed to remedy any potential bias that could add incorrect links to the oracle link set. Our strategies include three stages. We take the traceability between classes and sections as our example.

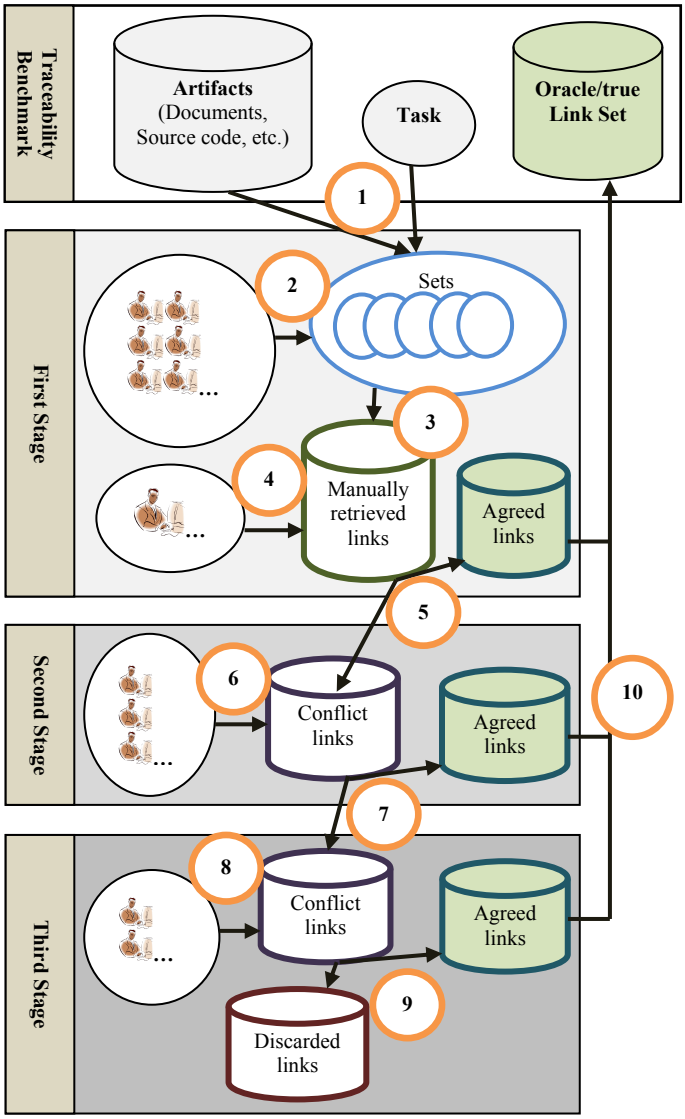

Figure 1. The strategy of oracle traceability link set development

At the first stage, the source artifact is divided into overlapping sets. (1). Each element of the source artifact can be assigned to two different sets. The number of sets depends on how much workload is assigned to each participant. The less the workload allocated to each participant, the smaller number of elements each set contains, and the more participants that need to be used. Next, according to the number of the sets, a group of participants needs to be recruited. These participants are required to have at least some knowledge about the selected project, called junior participants. Then every participant is allocated a set (2). They then manually identify links between the artifact in the set and another selected artifact based on the traceability rule mentioned above (3). In our example, we use classes as the source artifact and split all collected classes into 6 groups. 6 junior participants are then employed, each having a background of Java programming experience. Each is assigned a set and is required to find links between classes in the set and sections in the documentation based on the traceability rule.

After these participants complete their tasks, another group of participants with good understanding of the selected project, called senior participants, is recruited to verify these retrieved links based on the traceability rule (4). The link verification aims to identify the status of each retrieved link, whether or not it is correct, and to capture links that are missed by the previous participants. The number of participants still depends on how much workload is needed for each participant. If it is a reasonable workload for an analyst to verify all retrieved links, then using one senior analyst is enough. Otherwise, these retrieved links are split into overlapping sets. Every participant is responsible for one set. If participants all consider a link is true, then this link is in the set of agreed links, otherwise it is in the set of conflict links (5). For example, a class is assigned to two participants in the first group and two participants in the second group. If the four participants all identify the same link related to the class, then this link is considered as an agreed link. Otherwise, this link is considered as a conflict link as it is not unanimously recovered by the four participants. After the first stage, an agreed link set is generated. Each link in this set is agreed to be true by all participants who are allocated the source artifact of the link.

At the second stage, the set of conflict links generated at the first stage is randomly divided into overlapping sets, the number of which is based on how much workload each participant needs to undertake. A new group of participants are recruited based on the number of overlapping sets. Each of them is assigned a set to verify the conflict links by carefully studying the content in the selected artifacts (6). At this stage every conflict link is analyzed by two participants. If a link is considered to be a true link by the two participants, then this link is added to the set of agreed links, otherwise, it is a conflict link (7). In our example, every participant is required to carefully study the text of sections and the source code before making the decisions. After the second stage, an agreed link set is produced. Each link in this set is agreed to be true by two participants.

At the third stage, a senior analyst is employed to verify the set of conflict links produced at the second stage (8). This analyst carefully learns the content of the selected artifacts and also consults with another senior analyst to determine whether or not a conflict link is correct. In this example, the senior analyst needs to carefully learn the content of sections and the comments in classes. Each conflict link is analyzed by at least three participants, who have either identified the link at the first stage or have verified the link at the second or third stages. When three or more participants agree that a conflict link is not correct, this link is considered to be incorrect and is discarded; otherwise it is considered to be a true link (9). The third stage creates an agreed link set, in which every link is agreed to be true by at least three participants.

Finally, the three agreed link sets produced at the three stages are merged together to form the oracle traceability link set for the selected artifacts in the selected project. At the first stage, an agreed link is simultaneously captured by all participants who are allocated the source artifact of the link. At the second stage, an agreed link is unanimously considered to be true by two participants who have verified this link. At the third stage, an agreed link is agreed to be true by at least three participants who have identified or verified it in the three stages. They are stored in a file (e.g. a XML document, a table, a matrix etc.) to facilitate researchers to use and interpret them.

1) Probability of errors

After the establishment of the oracle link set for a project, an issue arises: what is the probability of errors (e.g. 1%, 5%, or 10% errors) in this oracle link set? To analyze this probability we build a formula to calculate the error probability based on the following three assumptions:

First, we assume that the probability of an error being made is dependent on the type of participant and stage. For example, we might make the following assumptions: 1) Junior analysts have an error probability of 20%, i.e. during link recovery, a junior analyst produces 20% incorrect links. 2) Senior analysts have an error probability of 10%. 3) If a senior analyst consults with another senior analyst, this senior analyst has an error probability of 5%; thus, he/she retrieves 5% incorrect links.

Second, we assume that links are independent. A link's recovery doesn't affect the probability of any other link's recovery. Moreover, the status (i.e. correct or incorrect) of any individual link doesn't affect the probability of recovering any other link and the probability of the status of any other retrieved link. For the sake of simplicity, we assume that every link has the same likelihood of being retrieved.

Third, we assume that errors made on links are independent; so an error made on any individual link does not affect the probabilities of errors on other links. Errors here include the errors of judging an actual correct link to be an incorrect link, judging an actual incorrect link to be a correct link, or failing to recover an actual correct link. The probability of making an error on a link does not influence the probability of an error made on any other link.

The oracle link set comprises three agreed link sets produced in the three stages. These link sets are independent. The probability of errors in the oracle link set is defined as:

$$\Pr[E] = \frac{\sum_{i=1}^{k}(n_i \times \Pr[e_i])}{N} \text{ , where}$$

N is the size of the oracle link set, i.e. the total number of links in the oracle link set. k is the total number of stages, here an oracle link set is established through three stages (k=3). n_i is the size of the agreed link set at the i^{th} stage, i.e. the number of agreed links at the i^{th} stage. $Pr[e_i]$ is the probability of errors in the agreed link set of the i^{th} stage. It is defined as:

$$\Pr[e_i] = \frac{\sum_{j=1}^{m_i}\Pr(x_j)}{m_i} \times \Pr(y_i) \text{ , where}$$

m_i is the number of participants at the i^{th} stage. $Pr(x_j)$ is the probability of incorrect links captured by the j^{th} participant at the i^{th} stage. $Pr(y_i)$ is the probability of participants simultaneously making the same mistake. For example, at the first stage, a class is assigned to three participants, all participants generate the same link for this class, and then this link goes to the agreed link set. We treat this link as a true link. But if this link is actually an incorrect link, $Pr(y_1)$ represents the probability of the three participants all making the same mistake of capturing the incorrect link at the same time at the first stage. Errors in $Pr(x_j)$ and $Pr(y_i)$ are distributed in the binomial distribution. We use the following binomial probability formula [22] to compute $Pr(x_j)$ and $Pr(y_i)$.

$$P(x) = \frac{n!}{(n-x)!\,x!} \times p^x \times q^{n-x} \text{ for } x = 0, 1, 2, \ldots, n, \text{ where}$$

n = number of trials. For $Pr(x_j)$, n = the number of agreed links retrieved by j^{th} participant at the i^{th} stage; For $Pr(y_i)$, n = the number of participants who identify a link in the agreed link set at the i^{th} stage.

x = number of successes among n trials. For $Pr(x_j)$, $x = x_j$ = the number of errors (e.g. 5% or 10% errors) in the agreed link set retrieved by j^{th} participant at the i^{th} stage; For $Pr(y_i)$, $x = y_i$ refers to at least x participants who consider a link is true at the i^{th} stage. In other words, x = the number of participants who identify or verify a link in the agreed link set at the i^{th} stage.

p = probability of success in any one trial. For $Pr(x_j)$, p = 0.2 (20%) for junior analysts, p = 0.1 (10%) for senior analysts, p = 0.05 (5%) for a senior analyst consulting another senior analyst. For $Pr(y_i)$, p = the average/mean error probability of n participants.

q = probability of failure in any one trial (q = 1 - p)

The error probability of links ($Pr[E]$) in the oracle link set depends on the error probability ($Pr[e_i]$) of links in each agreed link set generated at each stage. The error probability ($Pr[e_i]$) of links at each stage largely relies on each participant's error probability ($Pr(x_j)$) and the probability of several participants ($\geqslant y_i$) making the same mistake ($Pr(y_i)$).

E. Evaluation Metrics

After the development of the oracle link set, we can perform an evaluation and comparison of the traceability recovery technique under study to other techniques. The common metrics used in the evaluation of recovery techniques are precision, recall, and F-measure. These three metrics depend on three figures: correct links retrieved, incorrect links retrieved, and missing links. Correct links retrieved are those that are correctly captured by the recovery technique. Incorrect links are those that are wrongly detected by the recovery technique. Total links retrieved combines these two kinds of links. Links that are not found by the recovery technique are called missing links. Total correct links are the sum of correct links retrieved and missing links.

Precision can be defined as the ratio of the number of correct retrieved links over the total number of retrieved links. If precision equals 1, it means that all the recovered links are

correct, though there could be correct links that were not recovered.

Precision = Correct links retrieved / Total links retrieved

Recall is the ratio of the number of correct retrieved links over the total number of correct links. Recall = 1 indicates that all correct links are recovered, but there may be incorrect recovered links.

Recall = Correct links retrieved / Total correct links

The F-measure combines precision and recall based on their weighted harmonic mean to measure the effectiveness of retrieval. β is an adjustable weight to favor precision over recall. β=1 weights precision and recall equally. β=2 weights recall twice as much as precision. β=0.5 weights precision twice as much as recall.

F-measure = $((\beta^2+1)\text{Precision}\times\text{Recall}) / (\beta^2\text{Precision})+\text{Recall})$

Two sets of traceability links between selected artifacts are prepared in order to compute precision, recall, and F-measure. One set is produced by a traceability recovery system under evaluation. The other set is the oracle link set for the selected artifacts. The latter is critical as it is a crucial factor in determining the number of correct and missing links. Comparison of the two sets is then conducted to determine whether a link is correct, incorrect, or missing.

IV. Case Study

To validate the effectiveness of our approach, we have set up a case study to build a traceability benchmark for the evaluation of our traceability recovery technique introduced in [6]. This benchmark comprises four components: tasks, dataset, oracle/true traceability link set, and measures.

Tasks. Our particular recovery technique aims to capture traceability links between source code and documentation. Hence, the task is to trace links between code and documents.

Dataset. According to the task, the dataset is the collection of source code and documents. As our recovery technique [6] goes down to the class level in code and section level in documents, the dataset consists of classes and sections in documents. The system we used is JDK 1.5, a free open source software system for Java developers. We chose to use three packages (java.awt, javax.naming, and javax.print) from the JDK1.5 source code and their associated documentation. These three packages were chosen because of the availability and detail of corresponding natural language documents describing these parts of the system. Three documents [18] explain in detail the structure of packages. For example, JPS_PDF.pdf describes how the Java printing support works and which functions are implemented by which Java classes in the javax.print package. Table 1 describes the packages in JDK 1.5 and their corresponding PDF documents used in this study, as well as the number of Java classes and the number of sections in them. We divided these PDF files into sections based on their headings. For example, if a PDF document contains 10 headings, it is split into 10 sections; the contents of each are the text between one heading and the following one. We obtained 249 Java classes and 182 sections (or small documents). The traceability task becomes that of extracting relationships between these 182 sections and the 249 Java classes in JDK1.5.

Oracle/true traceability link set. In order to build the oracle traceability link set for JDK1.5, we recruited 11 analysts: 9 analysts had at least 6 years of Java programming experience, and 2 participants had more than 9 years of Java programming experience. We set the class artifact as the source artifact as classes are easier to be grouped than sections. We also set up two rules to assist participants in finding and verifying a link: 1) if a section directly mentions a class identifier or name, then this section is related to this class; 2) if a section describes tasks that a class should fulfill, then they are related.

TABLE I. JDK1.5 Packages and Documents [18]

JDK 1.5		**#classes/ sections**
Java packages	java.awt, javax.naming, and javax.print packages	249
PDF files	*JPS_PDF.pdf*: Java™ Print Service API User Guide	68
	dnd1.pdf: Drag and Drop subsystem for the Java Foundation Classes	41
	jndispi.pdf: Java Naming and Directory Interface™ Service Provider Interface(JNDI SPI)	73
	Total sections:	182

At the first stage, the classes were divided into 6 sets. 6 participants then manually retrieved links between sections in documents and classes by following the above two rules. After they completed their task, we asked a senior participant to conduct link verification that included verifying these retrieved links and capturing links missed by them. Then at the end of the first stage, 408 links were identified as conflict links, and 356 links were included in the agreed link set; each link was agreed to be true by two participants. At the second stage, 408 conflict links produced at the first stage were randomly divided into 3 overlapping sets. Three other participants verified these conflict links by carefully studying the text of documents and the comments inside code. Then at the end of the second stage, 75 conflict links were identified, and 333 links were included in the agreed link set; each link was verified to be true by two participants. At the third stage, we asked a senior participant to verify those links still having conflicts. This participant carefully studied the text of documents and the comments in code. This participant also consulted with another senior participant. Each conflict link was thus analyzed by at least 3 participants. When three or more reviewers agreed that the conflict link was a fault, we considered it to be an incorrect link and discarded it. Then at the end of the third stage, 4 links were considered as conflict links, and 71 links were added into the agreed link set; each link was identified to be true by at least three participants. The final oracle link set comprised 760 true links, which we then stored in a table. 110 out of 249 classes had no sections related to them.

Figure 2 shows the time taken by the six participants to manually capture links between 249 classes and 182 sections at the first stage. Participant 3 spent less time than others; 20 minutes taken to retrieve links. Participant 5 took the longest time (120 minutes) to capture links. Participant 6 used 90 minutes to perform the link recovery. Participants 1 and 2 each took 50 minutes, and participant 4, 40 minutes. On average, each participant spent around one hour to identify links

between 50 classes and 182 sections. Figure 3 shows the link recovery performance of each participant; namely, the percentage of links captured by each participant. Participant 3 retrieved the lowest number of links, 7%. Participant 5 recovered the highest number of links, 95%. Participant 2 recovered 80% of links. Participant 4 is 71%, 66% for participant 6, and 31% for participant 1. In total, the six participants retrieved 409 links and identified 136 classes with no related sections. Figures 2 and 3 show that participant 3 used the least amount of time but retrieved the lowest number of links, while participant 5 spent the longest time but captured the highest number of links. Participants commented that it was a tedious, boring, and time-consuming task. Moreover, one participant commented that he/she would favour automatic techniques to recover links.

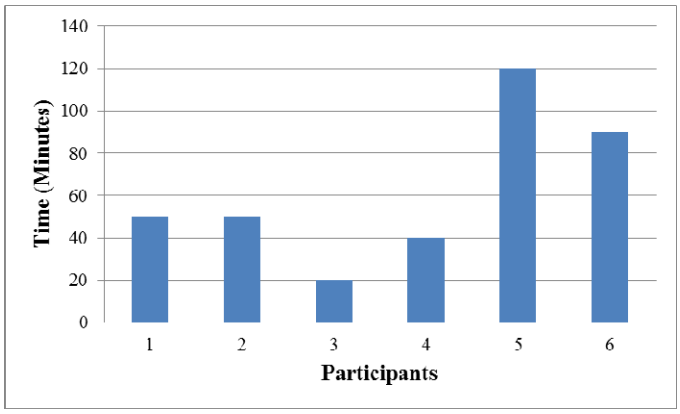

Figure 2. Time taken by the first six participants at the first stage

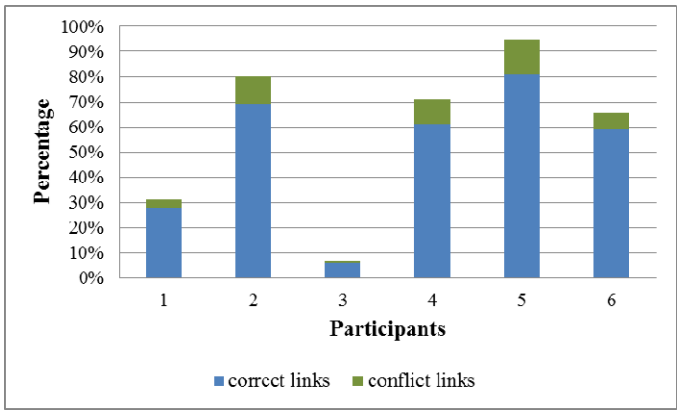

Figure 3. Percentages of links captured by the first six participants at the first stage (determining correct links vs. conflict links)

After the first six participants identified links, these retrieved links needed to be verified. Figure 4 shows the time taken by the remaining five participants to manually verify these retrieved links. The seventh participant verified links retrieved by the first six participants and captured links missed by them. This participant took 220 minutes to verify these retrieved links and to capture new links. This participant identified 408 conflict links that included 355 new links. Figure 3 shows the percentage of retrieved links identified as conflict by participant 7. Links retrieved by participant 5 contained 14% conflict links, 11% for participant 2, 10% for participant 4, 7% for participant 6, and 1% for participant 3.

The next three participants (from 8 to 10 in Figure 4) were asked to verify 408 conflict links generated at the first stage. Each participant at the second stage used around 87 minutes to verify these conflict links on average. They reduced the conflict links to 75. The last one (11 in Figure 4) verified these 75 conflict links. This participant took 180 minutes to undertake the link verification. At the end of the process, 4 links remained in conflict. On average, each participant took around two hours to verify links. By comparison with Figure 2, participants spent more time in the link verification than in the link recovery on average. Participants again commented that it was a tedious and very boring task. Four of them commented that it would be helpful to use traceability tools to support the link verification.

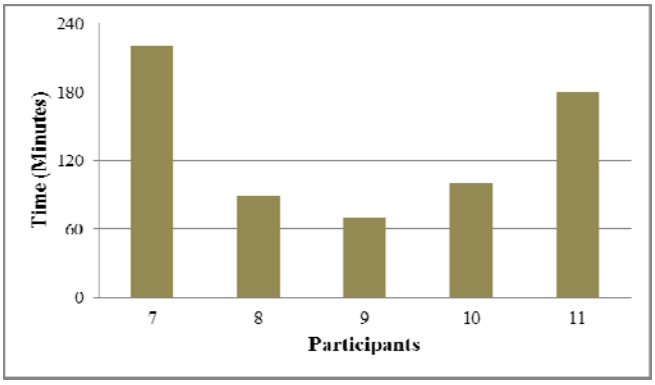

Figure 4. Time taken by the rest of the participants in verifying links

Measures. We use precision, recall and F-measure to evaluate the effectiveness of a traceability recovery technique. We used our new JDK1.5 benchmark to assess the performance of our combined retrieval traceability technique and toolset [6].

V. PROBABILITY OF ERRORS IN JDK1.5 BENCHMARK

The oracle link set for the JDK1.5 benchmark consists of 356 links retrieved at the first stage, 333 links verified at the second stage, and 71 links verified at the third stage. In order to compute the probability of errors in this link set, we need to calculate the probability of link recovery errors made by each participant ($Pr(x_j)$), and the probability of participants simultaneously making a same mistake of capturing an incorrect link ($Pr(y_i)$). Initially we used the example error rates for different participants introduced in section 3.4 (20% for juniors, 10% for seniors and 5% for seniors consulting one another) to demonstrate this method. Table 2 lists the results of $Pr(x_j)$ and $Pr(y_i)$ for having $\geq$ 10% link errors. We use STATDISK calculators to produce these results of $Pr(x_j)$ and $Pr(y_i)$. STATDISK can be free downloaded at http://www.statdisk.org. Then the probability of $\geqslant$ 10% link errors in the JDK1.5 oracle link set is calculated as follows:

$$\Pr[E] = \frac{\Pr[e_1]\times n_1 + \Pr[e_2]\times n_2 + \Pr[e_3]\times n_3}{N} = 0.027$$

for $E \geqslant N$*10%, where

- $N = 760$; n_i is the number of links in the agreed link set at the i^{th} stage, here $n_1 = 356$, $n_2 = 333$, and $n_3 = 71$;

$$\Pr[e_i] = \left(\sum_{j=1}^{m_i} \Pr(x_j)\right) \Big/ m_i \times \Pr(y_i)$$

- $Pr[e_1] * n_1 = 0.88122235 * 0.0225 * 356 = 7.05859$
- $Pr[e_2] * n_2 = 0.9998299 * 0.04 * 333 = 13.31773$
- $Pr[e_3] * n_3 = 0.0641971 * 0.0266119 * 71 = 0.121297$

We can use the same method to calculate the probability of $\geqslant$ 5% link errors in the JDK1.5 oracle link set:

$Pr[E] = 0.0292$ for $E \geqslant N*5\%$, where

- $Pr[e_1] * n_1 = 0.994423843 * 0.0225 * 356 = 7.965335$
- $Pr[e_2] * n_2 = 1.0 * 0.04 * 333 = 13.32$
- $Pr[e_3] * n_3 = 0.4771324 * 0.0266119 * 71 = 0.9015154$

TABLE II. PROBABILITY DISTRIBUTION FOR THE JDK1.5 ORACLE LINK SET BASED ON THE EXAMPLE ERROR RATES OF PARTICIPANTS

Stage	Participant	Retrieved Links (*R*)	Conflict Links (*C*)	Agreed Links (*n*=*R*-*C*)	Example Error Probability for Participant (*p*)	$Pr(x_i)$ for $x_i \geq n * 10\%$ (round to the nearest integer)
1st stage	1	10	1	9	0.2	0.8657823
	2	84	12	72	0.2	0.9940038
	3	18	2	16	0.2	0.8592625
	4	35	5	30	0.2	0.955821
	5	145	21	124	0.2	0.9990852
	6	117	12	105	0.2	0.9969877
	7	764	408	356	0.1	0.497614
	$\sum_{j=1}^{m_1} \Pr(x_j) \Big/ m_1 = \sum_{j=1}^{7} \Pr(x_j) \Big/ 7 = 0.881222357$					
	$Pr(y_1) = 0.0225$ for $y_1 \geq 2$, where n=2, p=(0.2+0.1)/2=0.15					
2nd stage	8	272	53	219	0.2	0.9999765
	9	272	123	149	0.2	0.9996381
	10	272	87	185	0.2	0.9998751
	$\sum_{j=1}^{m_2} \Pr(x_j) \Big/ m_2 = \sum_{j=1}^{3} \Pr(x_j) \Big/ 3 = 0.9998299$					
	$Pr(y_2) = 0.04$ for $y_2 \geq 2$, where n=2, p=(0.2+0.2)/2=0.2					
3rd stage	11	75	4	71	0.05	0.0641971
	$\sum_{j=1}^{m_3} \Pr(x_j) \Big/ m_3 = \sum_{j=1}^{1} \Pr(x_j) \Big/ 1 = 0.0641971$					
	$Pr(y_3) = 0.0266119$ for $y_3 \geq 3$, where n=5, p=(0.2+0.1+0.2+0.2+0.05)/5=0.15					

Although $Pr[E \geqslant N*5\%]$ is slightly larger than $Pr[E \geqslant N*10\%]$, the results are very close and small. Therefore, the probability of the oracle link set having $\geqslant$ 5% or 10% is very low, 0.0292 for $\geqslant$ 5% and 0.027 for $\geqslant$ 10% (all around 3%). In other words, the probability of building at least 95% correct links is very high (around 97%). The above calculation reveals three features. 1) The more senior a participant is, the lower probability of link errors the participant can make. For example, at the first stage, the first six participants are junior participants who have an error probability of 20%; their probabilities of making $\geqslant$ 10% link errors are larger than 85.9% (see Table 2). But the 7th participant, who is a senior analyst with an error probability of 10%, can achieve 49.8% probability of making $\geqslant$ 10% link errors. 2) The probability of participants simultaneously making the same mistake of identifying an incorrect links is very low, $Pr(y_1) = 0.0225$, $Pr(y_2) = 0.04$, and $Pr(y_3) = 0.0266$. This indicates that the chance of several participants ($\geqslant$ 2 or 3) retrieving an incorrect link at the same time is very rare. This reflects that our rigorous manual identification and verification strategies can largely reduce the probability of errors in the oracle link set. 3) The more participants that are allocated to verify a link, the lower the probability of link errors that can be achieved. For example, at the first stage, each link in the agreed link set was agreed to be true by two participants, $Pr(y_1) = 0.0225$ (see Table 2). If we add one more participant to verify a link and the third participant is a junior participant, then $Pr(y_1) = 0.0046$ for $y_1 \geqslant 3$, where n=3, p=(0.2+0.2+0.1)/3=0.16667. But if the third participant is a senior participant, then $Pr(y_1) = 0.0024$ for $y_1 \geqslant 3$, where n=3, p=(0.2+0.1+0.1)/3=0.1333.

Instead of using the assumed example error rates for participants we can make use of the observations and results obtained from the establishment of the JDK1.5 oracle link set to calculate the actual average error rates for the three types of participants. Table 3 shows the actual error rates for participants in the establishment of the JDK1.5 oracle link set. The actual error probability for a participant is computed as the number of retrieved links / the number of incorrect links, where retrieved links are links that are recovered or verified by the participant, incorrect links are links that are recovered or verified by the participant but are excluded from the final oracle link set. The actual error rate for a participant is an approximation as the number of incorrect links is obtained by comparison between links retrieved or verified by the participant and links in the final oracle link set hence may not be completely accurate. However it is a good approximation.

TABLE III. ACTUAL ERROR RATES FOR PARTICIPANTS DURING THE ESTABLISHMENT OF THE JDK1.5 ORACLE LINK SET

Stage	Participant	Retrieved Links (*R*)	Incorrect Links (*W*)	Actual Error Probability (*p*=*W*/*R*)
1st stage	1	10	1	0.1
	2	84	5	0.05952
	3	18	1	0.05556
	4	35	3	0.08571
	5	145	6	0.04138
	6	117	6	0.05128
	7	***764***	***31***	***0.04058***
2nd stage	8	272	3	0.01103
	9	272	8	0.02941
	10	272	6	0.02206
3rd stage	***11***	***75***	***2***	***0.02667***

We used nine junior participants: the first six participants at the first stage and three participants at the second stage. The average error probability for junior participants is 0.05066 (about 5%), which is much lower than the example error rate (20%) used above. The senior participant (the 7th participant at the first stage) has an error probability of 0.04058, which is also lower than the corresponding example error rate (10% or 0.1). The senior participant (the 11th participant at the third stage) who consulted another senior participant has an error

probability of 0.02667 (2.667%), which is lower than the corresponding example error rate (5%). We then apply the three actual error rates for participants to recalculate the probability of errors ($\geqslant$ 5% or 10% errors) in the JDK1.5 oracle link set.

We can use the same calculation method demonstrated above to compute the results of $Pr(x_j)$ and $Pr(y_i)$ for having $\geqslant$ 10% link errors based on the actual error rates for the three types of participants. The probability of $\geqslant$ 10% link errors in the JDK1.5 oracle link set is $Pr[E]$ = 1.2689e-4 for $E \geqslant N$*10%. This result shows that the probability of creating at least 90% accuracy in the JDK1.5 oracle link set is extremely close to 100%. The probability of $\geqslant$ 5% link errors in the JDK1.5 oracle link set is $Pr[E]$ = 0.0012 for $E \geqslant N$*5%. This result shows that the probability of making $\geqslant$ 5% link errors is 0.0012 (0.12%). In other words, the probability of building an oracle link set with accuracy of at least 95% is very high, around 99.9%. We thus conclude that our approach produces a high quality oracle link set.

VI. Cost-quality Tradeoffs

The most important part of establishing a benchmark is to create a high quality oracle link set (e.g. the link set with $\leqslant$ 5% errors). Building a high quality oracle link set depends on three factors. 1) The workload allocated to each participant. 2) The number of participants verifying a link. 3) The knowledge of the traced project of each participant, i.e. junior or senior.

In general, each participant at the same stage is allocated a similar workload. For the first stage, there are two groups of participants. The first group is to retrieve links between allocated artifacts. The second group is to verify links retrieved by the first group. For example, when we built the JDK1.5 oracle link set, the first six participants at the first stage captured links between allocated classes and documents. The 7th participant at the first stage verified these retrieved links. Every participant in the same group was assigned a similar workload at the first stage. The more workload that is assigned to a participant, the more effort they are required to make.

From the error probability calculation discussed above, we noticed that using a different number of participants to verify a link can affect the results of the probability of errors in the oracle link set. Using more participants to verify a link can produce a more accurate oracle link set. Using at least three senior participants to verify each link can achieve better results than using at least three junior participants or the combination of junior and senior participants, or at least two senior/junior participants. The participants' knowledge of the traced project can significantly affect the probability of errors in the agreed link set recovered by them. Based on the error probability calculation discussed above, and our observations in practice, we postulated then confirmed that the more senior a participant is, the lower the probability of link errors the participant will make and that these differences have a significant impact on the overall error probability.

Overall, if the workload assigned to each participant is certain, the best solution for building a high quality oracle link set is to recruit all senior participants, and to use at least three participants to verify a link. Unfortunately, it is very hard to recruit senior participants in practice. Moreover, it is not easy to decide how many times are appropriate to identify or verify links, which depends on the assigned workload. In our case, we assigned around 50 classes to each junior participant in the first group at the first stage. On average, each of them took around 60 minutes to identify links between 50 classes and the documents. Each link at the first and second stages was verified by only two participants. We still achieved 99.9% probability of producing at least 95% correct oracle link set based on the actual error rates for participants. Therefore, the alternative solution for building a high quality oracle link set is: 1) to use junior participants for the first group at the first stage and the group at the second stage; 2) to use senior participants for the second group at the first stage, because they not only verify links retrieved by the first group but also recover links missed by them; 3) to use senior participants at the third stage because they need to verify links that are still in the conflict link set after going through the two stages; 4) to use at least two participants to verify a link at each stage.

VII. Discussion

The actual probability of making $\geqslant$ 5% link errors in the JDK1.5 oracle link set is 0.12%. Our rigorous manual identification and verification strategies significantly improve the accuracy of the each stage's agreed link set. This evaluation illustrates that our approach can help researchers to develop a robust and high quality traceability benchmark to perform an evaluation and comparison of different recovery approaches.

However, our approach suffers from four problems that occur during the development of a traceability benchmark. 1) The difficulty of determining whether or not two elements in artifacts are in fact related. Although we provide a traceability rule to help in the identification of true links, we rely on participants' knowledge and understanding to capture links. This may lead to the capture of incorrect links. 2) How much workload is suitable for a participant to undertake? The more workload that is allocated to a participant, the more time and energy are required. Too much workload may make participants lose interest in participation. When we built the JDK1.5 benchmark, every participant took one hour to identify the related sections for 50 classes on average. But it took a longer time to do the link verification than the link recovery on average. 3) The difficulty in the recruitment. It is not easy to recruit a good number of participants who are required to have some knowledge of the selected project, especially for recruiting senior analysts. If a participant was new to the selected project, he/she might be more likely to capture incorrect links than someone who knows the project to some extent. 4) The scalability of benchmarks. Our approach is suitable to build benchmarks for projects of a reasonable size because of its approach of manually identifying and verifying links. But benchmarks produced by using our approach can be extended to include more elements, artifacts, tasks, and/or measures.

In future work, we will extend this benchmark to cover more classes and documents. This benchmark will then able to be used to evaluate tracing approaches and procedures for a wide range of tasks from different areas of software

engineering. We also will look at other probability distributions for the probability of incorrect links captured by each participant ($Pr(x_j)$) to cover the issue that links may have different probabilities of being retrieved. Because a link's recovery is highly dependent on the textual descriptions some links may be harder than others to find due to ambiguous wording issues. This can translate through increased error rates on particular links. The binominal distribution we have applied then may invalid as clustering may occur. However, using different probability distribution is unlikely to significantly affect the very low error rates in the oracle link set we have come up with. Because the probability of errors in the created oracle link set heavily depends on the probability of participants simultaneously making the same mistake ($Pr(y_i)$), which is very low.

VIII. Threats to Validity

The first threat to the validity of our approach is that false positive links may be included in the oracle link set. This is because a link agreed to be true by participants at each stage is put in the oracle link set even if it is actually incorrect. This can affect the accuracy of the actual error rate for each participant. Thus, it is important to expand our approach in the future by exploring how correct links should be defined and how to assist participants in identifying them. Second, some links may be harder to identify than others in practice. In that case, the binominal distribution used in our approach may not be suitable. Other probability distributions therefore need to be explored to cover this issue in the future. Third, the case we used is a small project that contains a small fraction of the source code and documents in the JDK1.5 system. It is not representative of large software systems. Our approach also may show different probability error results when applied to recover links between artifacts in other software systems by other groups of participants.

IX. Summary

We described a new approach to help researchers to establish affordable and robust traceability benchmarks. Our approach comprises five steps: task identification, artifact selection, project selection, oracle/true traceability link set development, and evaluation metrics. We designed rigorous identification and verification strategies to decide whether or not a link is true; every link is verified by at least two analysts. A benchmark for JDK1.5 was built by using our approach. We built a formula to compute the probability of errors in the created oracle link set. The probability of making $\geqslant$ 5% link errors in the JDK1.5 oracle link set is 0.12%. The accuracy of the agreed link set at each stage is significantly improved by our rigorous manual identification and verification strategies. The error probability results show that our approach can build a high quality oracle link set for the selected project.

We have made our new JDK1.5 benchmark public and we allow users to access or download it for free. Our benchmark is represented in a spreadsheet format. Anyone can review the data, apply it to evaluate their traceability approaches, and probably extend it to better meet their own needs. Users can download it from: http://tinyurl.com/7l3ohe4.

References

[1] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering traceability links between code and documentations," TSE, Vol. 28, No. 10, Oct. 2002, pp. 970-983

[2] G. Antoniol, G. Casazza, and A. Cimitile, "Traceability recovery by modelling programmer behavior," 7th WCRE, Nov. 2000, pp. 240-247

[3] A. Bacchelli, M. D'Ambros, M. Lanza, and R. Robbes, "Benchmarking lightweight techniques to link e-mails and source code," WCRE 2009, pp. 205-214

[4] A. Bacchelli, M. Lanza, and R. Robbes, "Linking E-mails and source code artifacts," ICSE'10, May 2010, pp.375-384

[5] E. B. Charrada, D. Caspar, C. Jeanneret, and M. Glinz, "Towards a benchmark for traceability," IWPSE-EVOL 11, Sep. 2011, pp. 21-30

[6] X. Chen and J. Grundy, "Improving automated documentation to code traceability by combining retrieval techniques", *26th ASE*, 2011, Lawrence, KS, pp. 223-232

[7] J. Cleland-Huang, A. Czauderna, A., Dekhtyar, O. Gotel, J. H. Hayes, E. Keenan, G. Leach, J. Maletic, D. Poshyvanyk, Y. Shin, A. Zisman, G. Antoniol, B. Berenbach, A. Egyed, and P. Maeder, "Grand challenges, benchmarks, and TraceLab: developing infrastructure for the software traceability research community," TEFSE 2011, May, Waikiki, USA

[8] J. Cleland-Huang, A. Dekhtyar, J.H. Hayes, G. Antoniol, B. Berenbach, A. Eyged, S. Ferguson, J. Maletic, and A. Zisman, "Grand challenges in traceability," Technical Report COET-GCT-06-01-0.9, Center of Excellence for Traceability, 2006

[9] A. Dekhytar, J.H. Hayes, and G. Antoniol, "Benchmarks for traceability?" TEFSE 2007, March, Lexington, KY

[10] O.C. Gotel, and A. C. W. Finkelstein, "An analysis of the requirements traceability problem," 1st RE, 1994, pp. 94-101

[11] J.H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: the study of methods," TSE, Vol. 32, No. 1, January 2006, pp. 4-19

[12] J.H. Hayes, A. Dekhtyar, S.K. Sundaram, E.A. Holbrook, S. Vadlamudi, and A. April, "Requirements Tracing On target (RETRO): improving software maintenance through traceability recovery," Innovations Syst Softw Eng (2007) 3, pp. 193-202

[13] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," TOSEM, 2007, Vol. 16 (4), Article 13

[14] A. Marcus, and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," 25th ICSE'03, 2003, pp. 125-135

[15] D. Oppenheimer, A. B. Brown, J. Traupman, P. Broadwell, and D. A. Patterson,. "Practical issues in dependability benchmarking. In Evaluating and Architecting System Dependability," October, 2002

[16] B. Ramesh, and M. Jarke, "Toward reference models of requirements traceability," IEEE Trans. Software Eng., 27(1), pp. 58-93, 2001

[17] J. Rilling, P. Charland, and R. Witte, "Traceability in Software Engineering—Past, Present and Future," CASCON Workshop, IBM Technical Report: TR-74-211, October 25, 2007

[18] PDF Version of JDK Documentation, 2010, Extracted on 10 Feb. 2010 from http://java.sun.com/j2se/1.5.0/download-pdf.html

[19] R., Seacord, D. Plakosh, and G. Lewis, Modernizing legacy systems: software technologies, engineering processes, and business practices. 2003, Addison-Wesley

[20] R. Settimi, J. Cleland-Huang, O. Ben Khadra, J. Mody, W. Lukasik, and C. DePalma, "Supporting software evolution through dynamically retrieving traces to UML artifacts," 7th IWPSE, 2004, Kyoto, pp. 49-54

[21] S. E. Sim, S. Easterbrook, and R.C. Holt, "Using benchmarking to advance research: a challenge to software engineering," ICSE 2003, pp. 74-83

[22] M. F. Triola, "Elementary statistics (7th ed.)". 1997, Addison Wesley Longman, Inc.

[23] X. Wang, G. Lai, and C. Liu, "Recovering relationships between documentation and source code based on the characteristics of software engineering," Electronic Notes in Theoretical Computer Science 243 (2009), Elsevier B. V., pp. 121-137