IEEE Access
practical innovations : open solutions

Received August 25, 2013, accepted September 17, 2013, date of publication October 22, 2013, date of current version October 31, 2013.

*Digital Object Identifier 10.1109/ACCESS.2013.2286822*

# A Study on the Effect of Traceability Links in Software Maintenance

**KHALED JABER[1], BONITA SHARIF[2], AND CHANG LIU[1] (Senior Member, IEEE)**

[1]School of Electrical Engineering and Computer Science, Ohio University, Athens, OH 45701, USA

[2]Department of Computer Science and Information Systems, Youngstown State University, Youngstown, OH 44555, USA

Corresponding author: C. Liu (liuc@ohio.edu)

**ABSTRACT** In software development, stakeholders of the same project often collaborate asynchronously through shared artifacts. A traceability system links a project's artifacts and therefore provides support for collaboration among stakeholders. Different stakeholders are interested in different types of traceability links. The literature often states that traceability is useful but expensive to build and maintain. However, there is no study showing reduction in effort when traceability links among various software artifacts are provided and used during the maintenance phase. This paper presents a study evaluating the benefits of using traceability among requirements, design, code, code inspections, builds, defects, and tests artifacts in the maintenance phase. Before the study, a survey was conducted at a large industrial firm to determine the type of links that different stakeholders are interested in. Twenty-five stakeholders from this firm participated in a survey to define the type of traceability links that were of interest to them. With this result, a traceability link model is proposed that categorizes different types of traceability links based on stakeholders' roles. This link model was used in the study. Twenty-eight subjects from industry and academia participated in the empirical study that was conducted after the survey. A prototype link-tracing tool, TraceLink, was developed and used in the study to present traceability links to the experimental group, whereas the control group was not given any links to solve the tasks. Five maintenance tasks were used in the study. The results show a significant improvement in task accuracy (86.06%) when traceability links were given to the subjects. In conclusion, a traceability model based on an industrial survey provided traceability views that are based on stakeholders' roles. This empirical study provides evidence that traceability links are effective in solving maintenance tasks with higher accuracy.

**INDEX TERMS** Software engineering, empirical study, traceability links, software maintenance.

## I. INTRODUCTION

Different stakeholders such as product managers, project managers, business analysts, developers, and testers are involved in the development of software systems. Product managers investigate, select, and develop products for an organization. They consider numerous factors such as the intended demographic, the products offered by the competition, and how well the product fits with the company's business model. Project managers are assigned by the performing organization to achieve project objectives. Business analysts or system engineers develop project requirements, developers implement the system's requirements and testers verify these requirements.

Software development is one of the most difficult tasks performed by humans. Because of the growing size and complexity of software systems, the coordination of work, artifacts, and developers has become a challenge [1]. During the development process of software systems, stakeholders develop different artifacts for the system. Some of these artifacts, for example, are project plans, requirement documents, testing plans, design documents, code, code inspection records, build information, and defect records. Different means of collaboration exist between stakeholders of a software system. One such means is *artifact-based collaboration.*

*Definition:* We define *Artifact-based collaboration* to be a form of asynchronous collaboration in which different stakeholders create or make use of artifacts at different points of the software development life cycle. There are two types of collaborations between the stakeholders of a software system. One is synchronous collaboration, which includes group meetings and other typical teamwork approaches. The other is asynchronous collaboration, where for example, a programmer works with an analyst to implement a certain feature. They do not meet and may not even communicate with each other directly. Artifact-based collaboration is accomplished through sharing and collaborative co-authoring of common

artifacts of a project. Artifact-based collaboration can be enhanced through a traceability system that explicitly links system artifacts together. A traceability system is comprised of a linked artifact space in which stakeholders traverse links between artifacts. The links in the traceability system are used for artifact-based collaborations, and they are referred to as traceability links in this paper.

The general consensus is that traceability between different artifacts helps reduce development and maintenance time and cost thereby improving the quality of the system [2]. However, there is no published evidence of the effect of traceability links in software maintenance tasks in industry. This paper seeks to gather such evidence through an empirical study. Before conducting the empirical study, a survey was conducted to determine the link model that fits stakeholder roles. This link model is used in the empirical study. Note that the intent of this paper is not to create a traceability system [3], rather the goal is to show the effect of links on maintenance tasks. For this reason, a simple prototype *TraceLink* was created that provided us with enough linking to run the study.

In the past decade, there has been a lot of progress made in the field of software traceability [4] (especially with link recovery and more recently with link evolution) in academia. The grand challenges document [5] for traceability lists specific research problems that need to be solved pertaining to traceability. Even though there has been progress made in academia, the same cannot be said about industry. Traceability mechanisms are still not widely adopted by industry due to the added effort needed to create and maintain traceability links between software artifacts [6] and the lack of supporting tools [7]. The goal of this paper is not to propose a new traceability system; but rather to investigate the following research questions resulting in two significant contributions to traceability evidence and future traceability tool developers:

- RQ1: In industry, what kinds of traceability links is each stakeholder interested in?
- RQ2: Do traceability links help software maintenance tasks?

In order to answer the first question, an industrial survey was conducted at a large software development firm. The results of the survey are presented from the point of view of each stakeholder. The second question is addressed by conducting a controlled experiment using industry experts as well as students in a senior software engineering capstone class.

This paper is organized as follows. The next section defines what we mean by traceability links. Section III presents the industrial survey, stakeholder benefits, real life scenarios that occurred at an industrial firm and a high-level traceability meta-model as a consequence of the survey. Section IV presents a controlled experiment conducted on the effect of traceability links (via a prototype linking tool *TraceLink*), on several software maintenance tasks. This is followed by a discussion in Section V and threats to validity in Section VII. Section VIII presents related work and Section IX discusses future work and concludes this paper.

## II. DEFINING TRACEABILITY LINKS

Software traceability refers to the process of creating/discovering and maintaining links between the different artifacts i.e., how does a particular source code element relate to a corresponding design element. The traceability definition used in this paper was built on previous work [8] in this area.

At the most general level, traceability links can be categorized into vertical and horizontal traceability links. Horizontal traceability refers to links within a model (intra-artifact links). Vertical traceability links are links across models. Under each of the two general link types we have causal links and non-causal links as defined in [8], [9]. The main points of causal, and non-causal links are reiterated in the definitions that follow.

*Definition: Causal Links* - Represent relationships that have an implied logical ordering between source and sink. For e.g., bug reports cannot be produced unless the source code is available. A causal link is always directional because it implies causality from source to sink; something happens and causes something else to happen. This establishes a partial ordering in time among the entities involved [10]. When this partial order in time is violated, it is possible that the link between the source and sink has been broken. Mathematically, the causal relationship is transitive, irreflexive, and anti-symmetric.

*Definition: Non-Causal Links* – Represent relationships that must agree with each other, but the causality cannot be clearly determined. For e.g., multiple versions of the same document in different languages must agree, but there need not be a causal relationship among them. A non-causal link is un-directional. Changes to the source or sink of a non-causal relationship can cause the logical semantics of the link to become invalid. Mathematically, the non-causal relationship is transitive, reflexive, and symmetric.

Most horizontal links are explicit, for example, two classes in a UML class diagram have a link since they have an association between them. But some links might exist within a model even though they are not explicitly linked. We call these hidden links i.e., links within a model that help in traceability. For e.g., three classes in a design model work towards a common feature of the system but there is no obvious linking between them on the UML class diagram. These links could help in feature location or impact analysis tasks. Such a link (or set of links) could be a causal or non-causal. We consider links across models and hidden links (within a model as well as across models) to be most important for traceability. Traceability link semantics such as those presented by Zisman et al. [11], [12] are viewed as complementary to the above definitions.

## III. INDUSTRIAL SURVEY

We studied the linking system utilized at a large industrial firm. The firm develops a set of artifacts, described below,

**TABLE 1.** Description of stakeholders interviewed.

| Stakeholders | Range of Experience (years) | Number |
|---|---|---|
| Developers | [5-10] | 7 |
| Testers | [3-5] | 9 |
| Project Managers | [7-10] | 4 |
| Product Managers | [5-10] | 3 |
| Business Analysts | [3-5] | 2 |

stored at different repositories. These repositories are utilized throughout the development cycle of software systems. The firm uses both the waterfall model and Scrum agile methods via the Rally tool[1] to develop their software systems.

The following are managed by the repositories utilized at the company:

- Project management artifacts such as the project plan, project charter, and statement of work.
- Requirements artifacts.
- Design artifacts such as use cases, sequence, class, state, and activity diagrams.
- Source code.
- Code Inspection records.
- Tests artifacts such as test cases and their execution results.
- Defects records.
- Builds and Releases artifacts contain builds and releases information.

At most established industrial firms and including the one used in this paper, the repositories that store the above artifacts are not linked together. Some of the above systems are purchased from different third party vendors. Companies that develop software projects are always in a need to link some or all of these repositories to help developers, testers, and project managers manage projects and meet deadlines.

In order to provide insight into the first research question, a survey was conducted at a large industrial firm. Three product managers (PdM), four project managers (PrM), two business analysts (BA), nine testers (T), and seven developers (D) were used in the survey to gain input into the types of traceability links they were interested in. Table 1 shows the number and range of experience among the stakeholders at the industrial firm. These stakeholders were asked to point out the type of links that would be helpful for them to perform their corresponding roles in the organization and the most important fields of each type of artifact that they are interested in Table 2 presents a form (excluding the x's) that was presented to each project stakeholder. It contains for each artifact the fields that belong to that artifact and cross references that links the artifact to other artifacts. Stakeholders pointed out the fields and the links that they are interested in by marking an 'x' near the specified row. These results are summarized by majority vote in Table 2. An 'x' refers to the most interesting fields to each stakeholder. The fields that are in bold represent fields that if changed, the appropriate stakeholder needs to be notified.

We summarize Table 2 below.

- *Requirements Artifacts:* Developers, testers, project managers, project managers, and business analysts are interested in the ID/version number, description, author, and date/version number fields.
- *Design Artifacts:* Developers are interested in ID/version number, name, description, author, date and status. Project managers are interested in status field.
- *Code Artifacts:* Developers are interested in the fields ID/version number, name, date, owner, and description.
- *Build Artifacts:* Developers and testers are interested in the fields ID/version number, location, and date. Project managers are interested in ID/version number and date.
- *Release Artifacts:* Developers and testers are interested in the fields ID/version number, location, date, and release notes. Product managers, project managers and business analysts are interested in the fields Id/version number, date, and release notes.
- *Test case artifacts:* Testers are interested in the fields ID/version number, title, name, test phase, status, last attempt, expected duration, actual duration, tester(s) names, expected results and obtained results in the test case artifact.
- *Project Management artifacts:* Project managers, product managers and business analysts are interested in ID/version number, name, author, date, and description fields.
- *Defect Artifacts:* Developers, testers and project managers are interested in ID/version number, description, create date, reported by, assigned to, status, attachments and logs/screenshots. Testers and project managers are also interested in severity, estimated hours, planned release, lifecycle fixed in, and actual hours fields.
- *Code inspection artifacts:* Developers are interested in ID/version number, project name, description, review date, participants, material, defect logs and status. Project managers are interested in ID/version number, project name and status.

There are two important findings based on this survey. *First*, the type of traceability links among system's artifacts is based on the role of the stakeholder. *Second*, the type of stakeholder role also determines which field a stakeholder is interested in. It is important to point out that one of the project

[1] http://www.rallydev.com/

**TABLE 2. Stakeholders' interest in types of information in each software artifact.**

| Requirements | Dev | Tester | PrM | PdM | BA |
|---|---|---|---|---|---|
| *Fields:* | | | | | |
| **ID/Version Number** | X | X | X | X | X |
| **Description** | X | X | X | X | X |
| Author | X | X | X | X | X |
| Date or version number | X | X | X | X | X |
| | | | | | |
| | | | | | |
| *Reference (Ref) to Test* | | X | X | | |
| *Ref to Design* | X | | X | | |
| *Ref to PM* | | | X | X | X |
| *Ref to Release* | X | X | X | X | X |
| **Design** | **Dev** | **Tester** | **PrM** | **PdM** | **BA** |
| *UML Diagram: Fields* | | | | | |
| Use case: **Id/Version no.**, Name, **Description,** Author, | X | | X (Status) | | |
| Class Diagram: **Id/Version no.**, Name, **Description,** Author, Date, **Status** | X | | X (Status) | | |
| Sequence Diagram: **Id/Version no.**, Name, **Description,** Author, Date, **Status** | X | | X (Status) | | |
| Collaboration Diagram: **Id/Version no.**, Name, **Description,** Author, Date, **Status** | X | | X (Status) | | |
| State Diagram: **Id/Version no.**, Name, **Description,** Author, Date, **Status** | X | | X (Status) | | |
| Activity Diagram: **Id/Version no.**, Name, **Description,** Author, Date, **Status** | X | | X (Status) | | |
| | | | | | |
| *Ref to code* | X | | X | | |
| *Ref to requirements* | X | | X | | |
| **Code** | **Dev** | **Tester** | **PrM** | **PdM** | **BA** |
| *Package (e.g., DLL), component or file Fields:* | | | | | |
| **ID/Version Number** | X | | | | |
| **Name** | X | | | | |
| Date | X | | | | |
| Owner | X | | | | |
| **Description** | X | | | | |
| | | | | | |
| *Ref to Design* | X | | X | | |
| *Ref to Code Inspection* | X | | X | | |
| *Ref to Build* | X | | X | | |
| *Ref to Defect* | X | | X | | |

| Build | Dev | Tester | PrM | PdM | BA |
|---|---|---|---|---|---|
| *Fields:* | | | | | |
| **ID/Version Number** | X | X | X | | |
| **Location** | X | X | | | |
| **Date** | X | X | X | | |
| | | | | | |
| *Ref to Release* | X | X | X | | |
| *Ref to Code* | X | | X | | |
| *Ref to Defect* | X | X | X | | |
| *Ref to Test* | | | X | | |
| **Test Case** | **Dev** | **Tester** | **PrM** | **PdM** | **BA** |
| *Fields:* | | | | | |
| **ID/Version Number** | | X | | | |
| Title | | X | | | |
| Name | | X | | | |
| Test Phase - Ex. Unit testing, system testing, etc. | | X | | | |
| **Status** | | X | | | |
| Last attempt | | X | | | |
| **Expected duration** | | X | | | |
| Actual duration | | X | | | |
| Tester (s) | | X | | | |
| Expected results | | X | | | |
| Results Obtained | | X | | | |
| Test procedure | | X | | | |
| | | | | | |
| *Ref Requirement* | | X | X | | |
| *Ref to Build* | | X | X | | |
| *Ref to Defect* | X | X | X | | |
| **Project Management (PM)** | **Dev** | **Tester** | **PrM** | **PdM** | **BA** |
| *Fields:* | | | | | |
| **ID/Version Number** | | | X | X | X |
| Name | | | X | X | X |
| Author | | | X | X | X |
| Date | | | X | X | X |
| Description | | | X | X | X |
| | | | | | |
| *Ref to Requirement* | | | X | X | X |
| *Ref to Release* | | | X | | |

| Defect | Dev | Tester | PrM | PdM | BA |
|---|---|---|---|---|---|
| *Fields:* | | | | | |
| **ID/Version Number** | X | X | X | | |
| Description | X | X | X | | |
| Creation date | X | X | X | | |
| Reported by | X | X | X | | |
| **Assigned to** | X | X | X | | |
| **Severity** | | X | X | | |
| **Estimated hours** | | X | X | | |
| **Planned release** | | X | X | | |
| Life Cycle Fixed In | | X | X | | |
| Actual hours | | X | X | | |
| **Status** | X | X | X | | |
| Attachments - logs/screenshcts | X | X | X | | |
| | | | | | |
| *Ref to Test* | X | X | X | | |
| *Ref to Build* | X | X | X | | |
| *Ref to code* | X | | X | | |
| **Code Inspection** | **Dev** | **Tester** | **PrM** | **PdM** | **BA** |
| *Fields:* | | | | | |
| **ID/Version Number** | X | | X | | |
| Project Name | X | | X | | |
| Basic Information: title, description, **review deadline** | X | | | | |
| Participants | X | | | | |
| **Material** | X | | | | |
| **defect logs** | X | | | | |
| **Status** | X | | X | | |
| | | | | | |
| *Ref to Code* | X | | X | | |
| **Release** | **Dev** | **Tester** | **PrM** | **PdM** | **BA** |
| *Fields:* | | | | | |
| **ID/Version Number** | X | X | X | X | X |
| **Location** | X | X | | | |
| **Date** | X | X | X | X | X |
| **Release Notes associated with the** | X | X | X | X | X |
| | | | | | |
| *Ref to PM* | X | | X | | |
| *Ref to Requirements* | X | X | X | X | X |
| *Ref to Build* | X | X | X | | |

managers at the firm mentioned that he is interested in all linking between repositories to show traceability when an audit occurs.

### A. TRACEABILITY META-MODEL

Based on the industrial survey, a traceability meta-model was developed. Different stakeholders in the software development cycle are interested in different traceability links between different artifact repositories. A traceability model must support traceability links that are specific to each stakeholder role. A meta-model that requires support for links specific to each stakeholder role is shown in Figure 1. In this meta-model, a *TraceModel* consists of one or more TraceLinks. A *TraceLink* contains two TraceElements (artifacts). A *TraceElement* has type and status attributes. The type is either ''source'' or ''target''. The status is set to ''modified'' when one or more of the important fields in *TraceElement* are modified. Stakeholder has a Stakeholder role that traces one or more specific TraceLinks. For example, a Stakeholder Role can be a project manager, a developer or a tester. Each of these stakeholders traces specific links based on their roles.

Using the meta-model, the overall traceability model for the industrial firm is reflected in Figure 2. The model when implemented will overcome some of the issues that have occurred at the firm due to lack of traceability. The links are defined based on stakeholders' roles. For example,

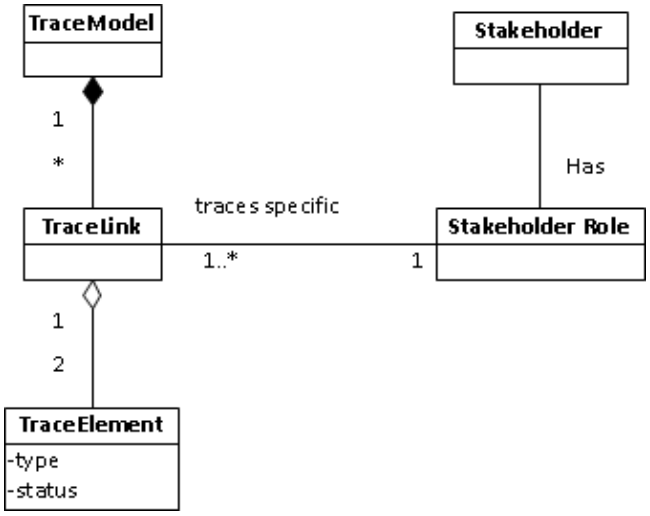


**FIGURE 1.** The traceability meta-model.

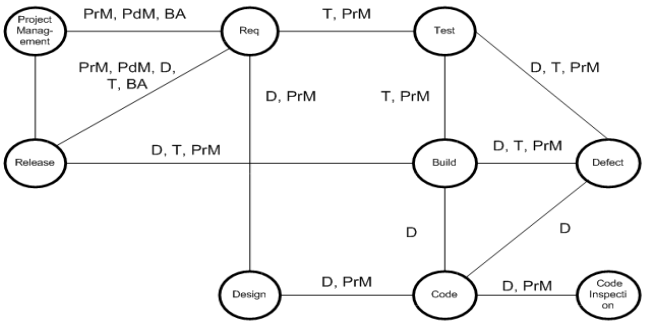


**FIGURE 2.** Five different views of traceability links for developers, testers, project managers, product managers, and business analysts. The edge names represent stakeholders interested in links between the two endpoints/artifacts.

using the model, a project manager can trace a requirement through design, code, code inspection, build, test, and defects. A developer can trace requirements to design, code, code inspection, build, and defects. The information in Figure 2 should be taken into account when developing a traceability tool.

The following summarizes the five views in Figure 2.

1) Product managers are interested in the links between these artifacts (project management, requirements) and (requirements, release).
2) Developers are interested in the following links (requirements, release), (requirements, design), design, code), (code, code inspection), (code, build), (code, defect), (release, build), (build, defect), and (defect, test).
3) Testers are interested in the following links (requirements, release), (requirement, test), (release, build), (build, defect), (build, test), and (test, defect).
4) Business analysts are interested in the links between these artifacts (project management, requirements) and (requirements, release).
5) Project manager is interested in the links between all artifacts.

While implementing the traceability model, reference fields for each artifact in the repository are used to point to another linked artifact in the same or external repository. To support the traceability across the artifacts utilized at the firm, all cross references shown in Table 2 need to be used. The traceability system shall support notifying stakeholders interested in the change. For example, stakeholders interested in an artifact change, register for the artifact. When the owner of the artifact modifies the artifact, he/she sets the status field in *TraceElement* to ''modified''. The system automatically checks for each artifact that has a status field set to modified and sends notifications to registered stakeholders. It is important to point out that the meta-model is used to link the project artifacts used in the study below. However, future work is needed to validate the model at the industrial firm and other firms as well.

### B. TRACEABILITY BENEFITS TO INDUSTRY – STAKEHOLDER VIEWS

We highlight below some of the traceability benefits reported by stakeholders at the firm. These benefits were captured as a result of interviewing different stakeholders listed in Table 1. They were asked to point out the benefits that they could achieve by using a traceability system that links all project artifacts. It is important to mention that these benefits reflect the interviewed stakeholders views. In the future, we plan to conduct a study to measure the actual benefits of using a traceability system for each of the stakeholders.

All stakeholders of a software system may benefit from utilizing traceability for collaboration. Product managers introduce new systems to organizations or add new functionality to existing systems. In both situations, they trace existing systems' artifacts such the project plan, project charter, statement of work, requirements, and releases. A traceability system should reduce the product manager's time.

Project managers use traceability through the development cycle of software systems. At the start of a project, project manager traces similar products artifacts to help him/her estimate time needed to create such artifacts. During the development of the project, project manager continues to trace the project artifacts to report status and checking for completeness. At the surveyed industrial firm, a senior project manager mentioned that he is interested in tracing all project's artifacts (i.e., tracing requirements through development and all the way through testing). Using a traceability system shall reduce the time spent by project managers tracing the project artifacts.

Business analysts also benefit from traceability. They can use traceability to reuse existing requirements or define new requirements for new systems. They trace existing artifacts such as the project charter and requirements. A traceability system reduces the business analyst's time in the development life cycle.

Developers are usually assigned one or more requirements to implement. Some of these requirements may be

changing or deleting existing functionality or developing a new functionality. In all of these situations, developers trace existing requirements, design, code, and test artifacts. A traceability system reduces developer's time in conducting these tasks.

Testers usually receive requirements and builds to test. The requirements may be already implemented in existing systems or they may be new. In both situations, they need to trace requirements, test cases and defects artifacts. A traceability system also reduces a tester's time.

Customers also benefit from traceability. For example, a developing organization may store their releases, upgrades, and their associated documentation in a system. Customers can access the system and trace what they have in the field (release builds) to what exists in the system. See scenario three in Section III-C for an example.

Another important activity that occurs in practice at industrial firms and benefits from traceability is auditing. Software systems are usually audited by external organizations to verify their quality compliance to ISO and Capability Maturity Model Integration levels. Auditors verify that all required documentation for the project are in place. The goal is to show the auditor the process in place, the artifacts that exist for the system and the traceability between these artifacts. From our experience, some findings that auditors see in practice are lack of traceability between system artifacts. The existence of a traceability system can help organizations pass audits and achieve a higher level of compliance for their software. This is particularly important in the medical domain since there are laws in place that require traceability.

### C. REAL-LIFE SCENARIOS

This section describes interesting scenarios that have occurred in real projects that have caused problems for the software firm interviewed in this study. They have occurred because there is no (visual/textual) traceability system that links the different repositories of the project and no automatic notification when artifacts in the repository changed. These scenarios were brought to light by conducting interviews during the survey with project managers, testers and developers.

*Scenario One*: A project was developed in which there was no traceability links between the project's builds and test. During the project, many internal builds are developed that needed to be tested by the testing team; the last build of the internal builds is the release build. The procedure utilized between developers and testers was that the developers load the internal build on a specific machine and notify testers of the new build through email. Sometimes developers forget to notify testers of new builds and the testers become aware of the build late in the testing cycle. Some builds were tested late and that caused the project's schedule to be delayed.

*Scenario Two*: A project was developed in which two organizations were involved in developing the project. The organizations use an XML file as an interface between them. One organization develops the file while the other uses it in one of its software components. The process used by organizations was that when a new version of the XML file is developed, the organization using the XML file is notified through email. Developers have reported that many new versions of the XML file were developed and the using organization weren't aware of them on time. Such situations affected the project's schedule.

*Scenario Three*: In another project, a device that has a network interface card (NIC) was developed. The NIC is utilized to get the internal data off of the device. Customers buy these devices and use them in their network. The purpose of the NIC is to present the device data to users. Sometimes device software is updated in which new data in the device is added or existing data is deleted. In this case, a new version of the NIC software is developed to pull the new data. In practice what happens is that customers update the device software but don't update the NIC software and vice versa. Here, linking is needed in which a customer can see a link to all versions of the software for both NIC and the device itself.

In each of the above scenarios, a traceability system that is well maintained would have reduced the time taken and mitigate the problems that occurred in communication, development and maintenance.

## IV. THE STUDY

The second research question seeks to determine if the presence of traceability links help in software maintenance tasks. This section presents details on the study conducted in industry and academia on the effect of traceability links in software maintenance tasks. The link model used in this study is based on the results of the industrial survey presented above. To conduct the study, we developed a prototype traceability tool that we call *TraceLink*. We first describe the prototype tool followed by the experiment details.

### A. TRACELINK

A prototype link tracing tool, *TraceLink* was developed that reflects the overall traceability model for the industrial firm as shown in Figure 2. See Figure 3 for a screenshot of the tool. The traceability links between the repositories are stored in an XML file. The ''Show Linked Artifacts'' button displays all artifacts that are linked to the one selected on the left pane in the window. In Figure 1, all artifacts linked to Req 2.1.1, for example, are displayed in the right pane. It is also able to show which repositories are linked.

When the user clicks the ''Check for Changed Artifacts'' button, the tool shows in the right pane all artifacts that have their status field set to ''modified''. For each of these artifacts, it lists the impacted artifacts that could be within the same repository or another repository. In the future, instead of users clicking the ''Check for Changed Artifacts'' button, the tool will automatically notify registered users of a modified artifact via email. Note that the goal of the study was not to determine if *TraceLink* was useful, rather it was to determine if the presence of traceability links (via the *TraceLink*) tool had an effect in solving the tasks. We use *TraceLink* as a

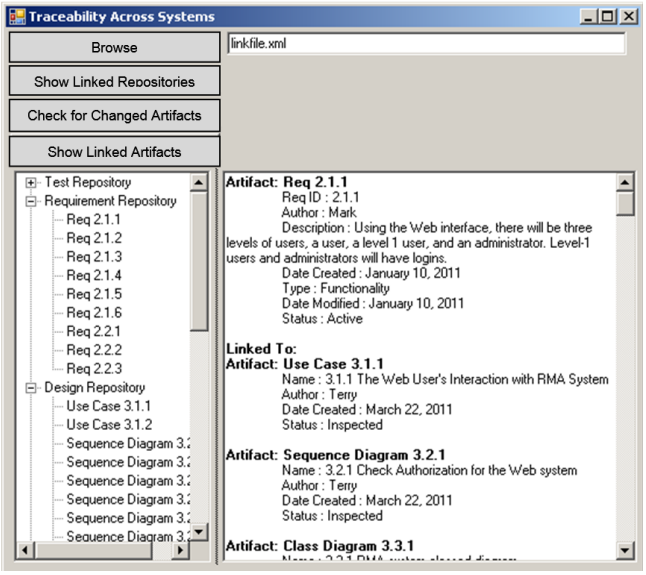


**FIGURE 3.** A snapshot of *TraceLink*.

means to view traceability links in the experimental group as described below in the experimental design.

### B. EXPERIMENTAL DESIGN AND HYPOTHESES

Following the template by Wohlin [13], the experiment seeks to *analyze* the effect of using traceability links via *TraceLink*, for performing software maintenance tasks *with respect to* effectiveness (accuracy) and efficiency (speed) *from the point of view of* the researcher *in the context of* industry practitioners and software engineering students. The detailed null hypotheses are given below. The alternative hypotheses are 1-tailed predicting *TraceLink* performs better. The hypothesis testing is done at 95% significance (alpha = 0.05).

$H_1$: There is no significant difference in task *accuracy* when performing maintenance tasks in the presence of using traceability links via *TraceLink* (versus not using it). $\mu$ (Accuracy$_{\text{with links}}$) = $\mu$ (Accuracy$_{\text{without links}}$).

$H_2$: There is no significant difference in task *speed* when performing maintenance tasks in the presence of using traceability links via *TraceLink* (versus not using it). $\mu$ (Speed$_{\text{with links}}$) = $\mu$ (Speed$_{\text{without links}}$).

$H_3$: Ability level does not significantly interact with *TraceLink* to have an effect on task accuracy, difficulty or speed.

The overview of the experiment is shown in Table 3. The main factor being analyzed is the link-tracing method with two treatments: without links (no *TraceLink*) and with links (links displayed using *TraceLink*). While analyzing the results we also looked at secondary factors such as the effects of subjects' ability level (high vs. low ability) on accuracy, difficulty level and speed combined with the link-tracing method. The difficulty level refers to a rating of easy, medium or hard given by the participant for each of the five software maintenance tasks.

### C. PARTICIPANTS

A total of 28 participants were involved in the study: 22 software engineering students and six practitioners (three developers and three testers) from industry. We split the participants into two groups. Only one group used *TraceLink* to solve the tasks. The other group were not provided with traceability links. To ensure a variety of abilities in both groups, the students were randomly placed into two groups. Industry participants were also randomly assigned into one of two groups of 14 people each. Both the control and experimental groups had eight high-ability subjects and six low-ability subjects. Six of the subjects from industry are very skilled practitioners who have been developing and testing software for at least 5 years.

### D. SUBJECT SYSTEM

A room management system (RMS) was used in the study. This system was developed by students in a software engineering capstone class and refined later by the authors of this paper to reflect a complete system. The system is about 10K lines of code. The system is responsible for managing shared office, meeting, laboratory, and teaching space and includes a web-based component and a physical device outside each room. The main features are reserving rooms, displaying reservations and editing reservations.

It is important to point out that the industrial survey provided two important outcomes: (1) the linking between project artifacts, for example, requirements is linked to test, test is linked to build, etc., and 2) five different stakeholders' traceability views. In this study, we used the first outcome. We plan to use the second outcome in another study as part of our future work.

A total of 165 artifacts were part of the test, requirements, design, code, code inspection, build and defects repositories. Project management and release artifacts are not included in the study because there were not available for the RMS at the time we conducted the study. We manually traced 402 links between artifacts based on the results of the industrial survey (See Figures 1 and 2). These links are verified with industrial stakeholders. Table 4 shows the different links from source to target between the repositories in our system.

### E. TASKS

There were five tasks created for the RMS each under a different category. Task 1 (T1), was about adding new functionality. Task 2 (T2), consisted of changing existing functionality. Task 3 (T3), dealt with fixing incorrect behavior. Task 4 (T4), was responsible for porting existing functionality and Task 5 (T5), dealt with an IDE change. For example T1 was based on adding new functionality: *Customer requested the following new requirement for the RMA system: 2.1.8. For some rooms, the maximum time of reservation is 2 hours for any user. What are the impacted artifacts due to the new requirement?* All tasks, T1, T2, T3, T4, and T5 require tracing

**TABLE 3.** Experiment overview.

| | |
|---|---|
| **Goal** | Study the effect of traceability links in software maintenance tasks |
| **Independent variables** | Tracing method: with links (using TraceLink), without links |
| **Dependent variables** | Accuracy, speed, difficulty level |
| **Secondary factors** | Ability level (high, low)<br>Type (industry, academia) |

**TABLE 4.** Number of links in the Room Management System. The number of artifacts in each repository are Test (113), Req (9), Design (19), Code (8), Code Inspection (2), Build (2), Defects (12).

| Source | Target | # Links |
|---|---|---|
| Test | Build | 113 |
| Test | Defect | 12 |
| Requirements | Test | 111 |
| Requirements | Design | 39 |
| Design | Code | 83 |
| Code | Code Inspection | 16 |
| Code | Build | 16 |
| Code | Defects | 12 |
| Total | | 402 |

**TABLE 5.** Tasks used in the study.

| ID | Category | Task Text |
|---|---|---|
| **T1** | Adding new functionality | Customer requested the following new requirement for the RMA system. What are the impacted artifacts due to the new requirement?<br>2.1.8 For some rooms, the maximum time of reservation is 2 hours for any user. |
| **T2** | Changing existing functionality | Customer requested a change to Requirement 2.1.2; the following is the modified requirement. What are the impacted artifacts due the new change?<br>2.1.2: The Web interface shall display the status of the room. If the room is currently occupied, the occupant decides whether to display his/her name. The time when the meeting will finish must be displayed. The Web interface shall display occupied and available status by a red or green light. |
| **T3** | Fixing incorrect behavior | Customer has found the following defect in which a user (not a level-1 user) is allowed to release a reserved room. A developer has investigated this defect and found that the reservation class code has a bug. What are the impacted artifacts due to this defect? |
| **T4** | Porting existing functionality | The development of a new system is starting, and the functionality of creating and deleting logins to level-1 users supported by the RMA system needs to be ported to the new system. What are the impacted artifacts? |
| **T5** | IDE change | Microsoft Studio version 2008 was used to develop the Room Management Appliance system (RMAS). A new version of Microsoft Studio version 2010 is released. Due to the new release, the RMAS is recompiled and a new build 6.3 is created. What are the impacted artifacts? |

the impacted artifacts for RMS. The order in which task should be solved first by participants does not matter. There is no dependency between the tasks. Table 5 shows the tasks used in the study. We have provided all the study material at *http://www.csis.ysu.edu/~bsharif/tracelinkstudy* and refer the reader there for more details.

### *F. RUNNING THE STUDY*

A week before the study was conducted the participants were given a document describing the subject system. The document contained a description of all the repositories set up for the system as well as information within each repository such as requirements and design documents. This was done to familiarize the participants with the system. The students were also given a quiz-like assignment to make sure they read the materials ahead of time. They were also given a 15 minute lecture on the importance of software traceability and the problems/challenges facing industry.

The study was conducted at three different locations (classroom, computer lab, industrial firm) and took approximately one hour. The tasks were presented to the subjects on paper. They were asked to trace the artifacts that were impacted for each software maintenance task. Subjects were asked to note the start and end time for each task on paper as well as indicate the level of difficulty (Easy, Average, Difficult) they faced for each of the five tasks. All the data was collected on paper questionnaires and later tabulated.

In addition to the tasks provided to all, people in group 1 (that were provided no links) were provided with an electronic

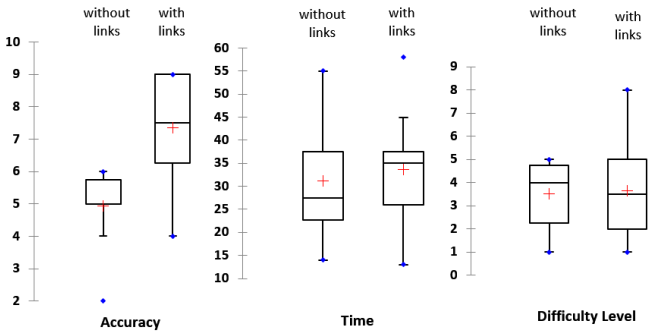


**FIGURE 4. Descriptive statistics for the three dependent variables (all data).**

file that contained all artifacts for the system. People in group 2 were provided with the same material as group 1, and in addition they were also presented with traceability links between the artifacts via the TraceLink tool. The tool contains all artifacts and the 402 links listed in Table 4. A sample demo for group 2 on how to use the tool was given before the study began. The academia subjects took the study in a computer lab while industry subjects did the study at their workplace. After all subjects were done with the five tasks, they completed a post-questionnaire.

### G. EXPERIMENTAL RESULTS AND ANALYSIS

Due to non-normality of the data (based on results from the Shapiro-Wilk test) and a sample size less than 30, we use the non-parametric Mann-Whitney test (for non-paired samples) to determine statistical significance between the two groups. Figure 4 shows boxplots for the three dependent variables: accuracy, speed (time), and difficulty level.

#### 1) ACCURACY, SPEED, DIFFICULTY, AND INTERACTION EFFECTS

Each question was given a score based on the correct answer given by subjects. The score denotes the accuracy. Table 6 shows the p-values for the Mann-Whitney test for each of the five tasks as well as all the tasks accumulatively. With respect to all the tasks (last row), we see that the only significant variable is accuracy. We are thus able to reject the null hypothesis $H_1$, showing that there is a significant difference (p-value = 0.0001) in task accuracy when performing maintenance tasks in the presence of traceability links presented via *TraceLink*, resulting in higher accuracy. T4 seemed to benefit the most from the presence of traceability links via *TraceLink* (p-value = 0.0003) followed by T5 and T2. No significance was detected in T1 and T3. One reason could be due to the fact that T4, T5, and T2 were harder than T1 and T3. When all subjects are considered, the users that were presented traceability links in *TraceLink* were 86.06% more accurate than subjects that didn't use traceability links.

With respect to speed and difficulty, the only time we achieve significance is with T5. Thus we cannot reject $H_2$ which states that there is no difference in speed between the group presented with traceability links via *TraceLink* vs. the control group. It is important to mention that subjects using *TraceLink* pointed out that they didn't have enough training on using the tool before performing the tasks of the study and that impacted their speed. They said that they were learning how to use the tool while they were solving the tasks. This may be one of the reasons why we did not see more improvements in speed. Not enough training was given to subjects due to their limited time. A future replication of the experiment with more training would help determine if this was the case. It is important to mention that industry subjects were 21% faster in solving the tasks using traceability links than their colleagues that didn't use the tool that showed them traceability links in the system.

In order to determine interaction effects, a linear mixed-effects regression model is fit to each of the dependent variables: accuracy, speed, and difficulty level with respect to all tasks accumulatively. The explanatory variables were Group (with links, without links), Ability (high, low) and Type (industry, academia). Table 7 shows the model parameters for accuracy and time only, because difficulty level was not significant for any explanatory variable.

As we can see from Table 7, none of the interactions are significant. We do find that the presences of traceability links had a significant effect on the accuracy variable (p-value = 0.009), which confirms the results from the Mann-Whitney test presented earlier. Ability (high, low) and Type (academia, industry) are individually significant with respect to the time dependent variable. Based on the observations, we cannot reject the null hypothesis $H_3$ which states that Ability does not interact with the link tracing method to have an effect on time or accuracy.

There are some interesting observations to be made despite not finding any significant interaction effects. With respect to accuracy, there was a very small difference between Type (academia and industry) within the group that used traceability links via the tool. There was also a small difference between Ability (high and low) within the experimental group compared to the control group (presented with no links or tool). Both high and low ability subjects scored higher in the experimental group vs. the control group.

With respect to time, in the experimental group, students and industry subjects were also comparable (See Figure 5). Without traceability links, industry subjects took longer than students. This can be attributed to the fact that the subjects from industry took the study more seriously. If students did not get an answer within a certain time they just moved on to the next task. However, if they were given traceability links via the tool, they tend to find the task more solvable and spend time on it. Another observation is that subjects with high ability levels took longer than those with low ability levels, again due to the seriousness of the subjects wanting to perform well.

#### 2) POST-QUESTIONNAIRE RESULTS

The post questionnaire asked the participants to rate their experience in the study. Both groups were given a common

**TABLE 6.** Resulting p-values of the Mann-Whitney test (one-tailed).

| Task | Accuracy | Speed | Difficulty |
|---|---|---|---|
| T1 | 0.666 | 0.877 | 0.988 |
| T2 | 0.002 * | 0.089 | 0.143 |
| T3 | 0.153 | 0.997 | 0.953 |
| T4 | 0.0003 * | 0.201 | 0.286 |
| T5 | 0.007 * | 0.016 * | 0.014 * |
| All | 0.0001 * | 0.759 | 0.527 |

**TABLE 7.** Complex model for *accuracy* and *speed* dependent variables for all tasks. The first line indicates accuracy with speed parameters given in parentheses.

| Variable | Value | Standard Error | p-value |
|---|---|---|---|
| Intercept | 5.000<br>(31) | 0.650<br>(3.801) | (<0.0001) *<br>(<0.0001) * |
| Type<br>(Academia vs. industry) | -1.000<br>(17.667) | 1.062<br>(6.208) | 0.354<br>(0.008) * |
| Group<br>(with links vs. without links) | 3.333<br>(-10.333) | 1.187<br>(6.940) | 0.009 *<br>(0.148) |
| Ability<br>(High vs. low) | -0.067<br>(-11.433) | 0.881<br>(5.147) | 0.940<br>(0.035) * |
| Type * Group | -0.933<br>(17.933) | 1.502<br>(8.779) | 0.539<br>(0.051) |
| Group * Ability | 0.400<br>(3.100) | 1.245<br>(7.279) | 0.750<br>(0.673) |

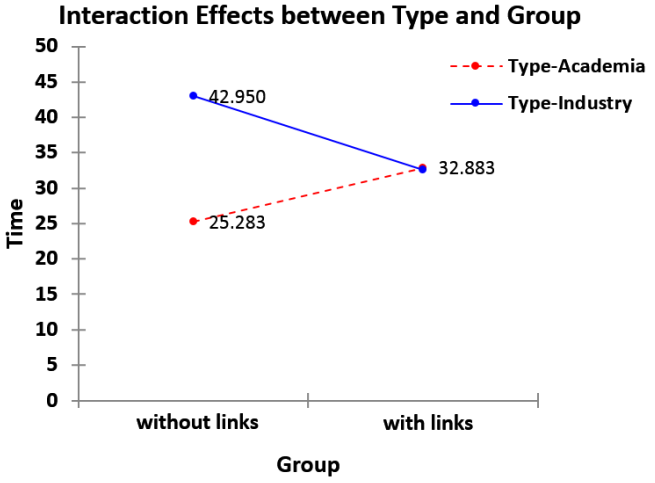


**FIGURE 5.** Interaction effects between the Type (academia vs. industry) and Group.

set of 7 questions. The questions were based on time needed, document understanding, clear objectives, clear tasks, diagram reading difficulty, task difficulty overall, and percentage of time spent reading the document. The group that was given traceability links answered four additional questions regarding percentage of time spent using the traceability links in the tool, tool usefulness, tool user friendliness and general comments on improving the way links are presented to them. On average, the experimental group (given the links) felt that they had more time to complete the task than the control group without links. The requirements document and tasks were clear to all subjects. The Mann-Whitney test found a significant difference between the experimental and control groups when it came to percentage of time spent reading the document vs. using traceability links. The group that used the tool reported that the tool was very useful and easy to use. The percentage of time taken using the tool was 60% on average.

The industry participants were given an additional post-questionnaire to fill out and were also interviewed after the study. The questions were open ended to probe them with respect to their experience with maintenance tasks that benefit from traceability in their daily work as well as potential features of a traceability link presentation tool (discussed in the next section). Some of the important highlights are presented below.

- Four out of five developers acknowledged that they encountered the situations described in the software maintenance tasks in their daily work. This suggests that the tasks developed for the study were similar to real-world tasks developers encounter in industrial software.
- Two out of five developers reported that they like to see links directly between requirements and source code since design information is usually out of date as soon as a line of source code is added/modified.
- Four out of five developers reported that visual linking is very helpful. One of the four developers reported that

providing visual linking for small size projects would not be that useful, but as the project size increases, its importance would also increase.
- Another developer reported that visual linking might help out in small projects, but when used with a large project, the view will be a large birds-nest picture which would be impossible to decipher. This shows that even though the developer appreciates traceability, he/she does not want to be overburdened with a hodgepodge of links.
- One of the developers reported that there is a need for required fields for the artifact in each repository that helps linking the repositories.
- Another developer stated that it would be nice if an item has tags (topics) and search functionality using tags. This was a very important observation since search trumps navigation.

## V. DISCUSSION

The results of the study indicate that using traceability links leads to significantly more correct answers for software maintenance tasks. With respect to speed, industry subjects who given traceability links (via *TraceLink*) were faster than subjects that were not given any links (group with no *TraceLink*). In academia, however, subjects who did not have traceability links were faster. This can be due to the fact that industry subjects are more accustomed to using tools (given that the links were presented via the *TraceLink* tool) compared to students; in addition, more training was needed on the tool as pointed out by some subjects. With more training we could expect to see a significant difference in time needed to solve the tasks. This is left as a future exercise.

Results also show that the use of traceability links reduces the gap between high and low ability subjects as well as between academia and industry subjects. This is an important finding because given traceability links, we can have novices solve tasks faster and hopefully improve their skills that lead them to become experts in the future.

Note that the study conducted in this paper tries to determine the effect of the presence of traceability links in software maintenance tasks. The method of how the links are presented to the user is of lesser importance in the study. We chose to have the links presented in a prototype tool called *TraceLink*.

The observations presented in the rest of this section are useful to traceability tool developers in order to get more industry acceptance of the tools they develop to present traceability links to human analysts/developers. It also incorporates the different views from different stakeholders; another contribution of this paper. The following are interesting findings as stated by developers. One developer stated ''I would only expect it to list the connections in a simple-to-use way making them easier to understand''. This implies that traceability needs to be effortless and almost second nature in order for it to be adopted. Another developer stated that ''as an automated tester, all test cases are automatically generated. So I don't need to trace as much as the questions in the study''. The automated tester believes that when creating the automated test suite, traceability is required, but after that tracing is not needed much. This could be the case in some instances.

One developer stated the following: ''The developer is the one who is generating most of the information contained in these repositories. The design and requirements in many organizations are generally not well defined, and therefore do not really provide good information. At best, that information is helpful at the beginning of a development project, but are soon outdated because they are not kept up-to date. A project manager or test engineer would use the tools in a different manner and seek different information than a developer, but they are generally light users of the tools and generally are manipulating the data instead of generating the information''.

## VI. DEVELOPER COMMENTS ON TRACELINK

This section presents what developers are interested to see in a traceability tool to help them in software maintenance tasks. Based on the input from developers in industry, some of the features that need to be supported by a traceability system are given below. These were all additional feature requests made by the subjects as they used *TraceLink* to solve the maintenance tasks. These observations are mentioned here with the goal of aiding future traceability tool developers.

- A simple search function similar to *grep* would be required. Note that *TraceLink* provided a way to find all links that were related to a given artifact however searching on specific keywords was not implemented.
- A traceability system should have more than one level of linking. For example, if artifact A is linked to artifact B and B is linked to artifact C, the tool shall support A to B and A to C linking. This is important during requirements gathering to test for contradicting requirements.
- A traceability system should support functionality to create different kinds of reports. For example, the tool should provide a table format. A report can be a table that lists builds, associated requirements, defects, test cases, etc.
- A traceability system should support artifact linkage in a graphical format in which filtering is a must to avoid being presented with an overwhelming number of links.
- Traceability needs to be almost effortless and blend in with the software process in order for it to be adopted in industry.
- All artifacts associated with a project should be linked to the project plan and provide easy navigation. When a developer gets assigned to work on a project, the developer sometimes reviews the project plan. The project plan mentions requirements, design, code, test, and other artifacts. It will be nice if the project plan has links that user can click on to get to the requirements, test cases, and other artifacts associated with the project. This type of traceability is not only important to developers, but also to project managers.

- A traceability system should provide views that are specific to stakeholders. This is an important feature that tends to be overlooked. For example, a developer who needs to visit the links that are of interest to him/her doesn't need to dive into an overall view that has links between all artifacts that is most useful to the project manager. Too many links impact developers' time and accuracy because the view can get complex and errors may occur.

The abovementioned features are useful to traceability tool developers in order to have their tools more readily adopted by industry. The lesser the effort required by a stakeholder to use a traceability system, the easier it is to adopt. Note that *TraceLink* was not designed to provide a fully working traceability system, rather an electronic way of presenting subjects with traceability links.

## VII. THREATS TO VALIDITY

The four main threats to validity are internal, external, construct and conclusion validity. With respect to internal validity, the study was between-subjects and did not suffer from learning effects between groups. Within each group, each task was from a different category and didn't overlap so there were no learning effects involved there either. The study was done in a classroom/lab setting or an industrial setting with dedicated time allotted to it, so as to minimize any other external factors that might have an effect on the results. A tutorial was given to all subjects prior to the study to make sure all subjects were at the same knowledge level we expected. None of the subjects were familiar with the room management system prior to the study.

With respect to construct validity, we choose to measure accuracy, speed, and difficulty level. Since our study goal was to assess the effect of traceability links on software maintenance tasks, we measure performance on the tasks via accuracy and speed. Better performance indicates higher accuracy and lower speed i.e., time to complete task. The difficulty level gave us some indication of how difficult the tasks were for them to solve. This combined with accuracy and speed was used in the analysis. The students were not told and did not know the hypotheses in advance. Some of the project managers were developers before and their input may be affected by their previous role. It would be nice to have the same number of industry experts as students but it was hard to find many developers volunteering to invest their time for this activity.

External validity deals with generalizing the results of the study. The study and survey is conducted at one industrial firm and on one system. The survey does reflect a single firm view. However, it is a step towards identifying links interesting to different stakeholders. It is quite challenging to find a bunch of firms willing to participate and invest time in surveys and studies such as the one we conducted. We welcome traceability researchers to build on top of the results in this paper and conduct such surveys with other industrial firms to advance this body of knowledge. Replicating this study with a larger sample from different organizations in different domains and with different systems and tasks can help reduce this threat. We leave this as future work. Even though the subject system used is not a large open source system, the questions asked with respect to a feature that needs to be added/changed is comparable to similar issues in open source issue tracking systems. We derived our tasks after browsing the Bugzilla database for various projects. The developers also agreed in the post questionnaire that the tasks were realistic. Also, there are not many open source systems that have traceability matrices available for use which makes the choice of a subject system very narrow. Most traceability papers use the same academic datasets such as iTrust [14] to validate their link retrieval/evolution algorithms. In this regard, we can consider the room management system dataset as yet another dataset with developer provided traceability links that could be used by traceability researchers in the future. Our results from the study are only generalizable to maintenance tasks that are similar (i.e., finding artifacts that need to be changed if a feature is added, changed, or behaves incorrectly). Our results from the industrial survey are generalizable only to industrial firms that follow similar process models. We do not claim that these results would apply to all settings. Further empirical studies are needed to validate this generalization.

With respect to conclusion validity, due to low sample size and non-normality of data we use the Mann-Whitney test for hypotheses testing.

## VIII. RELATED WORK

This section first presents related work in software collaborations. Hildenbrand et al. [15] evaluates collaborative approaches in requirement engineering, design and modeling processes, implementation, testing and maintenance. Whitehead [16] presents a list of goals for software engineering collaboration and surveys existing collaboration support tools in software engineering. He divides the collaboration tools into four categories: model-based, process support, awareness, and infrastructure. Trude et al. [1] presents empirical studies on collaborative software development. The studies are on the areas of collaboration during requirement engineering, collaboration during design, collaboration on distributed software development, existing collaboration tools, new collaboration tools, collaboration in open source, and collaboration in project management. Different tools are identified in [1], [15], [16] for collaborative software development. Some of these tools are configuration management tools [17]–[19], instant messaging tools [20], [21], source code annotation tools [22], [23], processes tools [24], awareness tools, [25], [26] and project management tools [27], [28]. None of these studies focus on artifact-based collaborations via traceability: a view we take in this paper.

We now present representative related work in traceability, models, tools, link recovery, link evolution, link visualization, empirical studies, and case studies in industry. We also describe how this work complements, differs and adds to the current understanding of the community. In our prior

work, traceability link definitions and a link model was developed [8]. A fine grained differencing mechanism to evolve traceability links was proposed in [29]. Spanoudakis and Zisman [11], [12] describe types of traceability links between requirements, use cases and analysis object models. Their links are based on existing software systems and also take into account software product lines. The link types presented in [11], [12] can be used in the model presented here targeted towards stakeholder roles.

Mäder et al. [30] conducted a study similar to the one presented in this paper. It assesses the effect of requirements traceability for maintenance tasks but uses only academic subjects, whereas our study uses subjects from academia and industry. Besides differences in the subject pool, they only trace requirements to code. Our study assesses traceability between requirements, design, code, builds, defects and test cases and supports many stakeholder roles. Similarities are that both studies use a prototype tool to test the effect of traceability on maintenance tasks, albeit with different subject pools. Both studies find that traceability links help accuracy and efficiency of maintenance tasks. In addition, our study finds that industry subjects who used traceability links (via the *TraceLink* tool) perform a lot better compared to their colleagues who didn't use any traceability links. Recall that in our study, traceability links were only given to one group and they used the links via the prototype *TraceLink* tool we built.

Traceability link recovery is an important part of any traceability system since it supports existing systems that don't have links defined a priori. DeLucia et al. discuss managing traceability for impact analysis and present the major challenges and research directions [31]. There are many papers such as [32]–[37] addressing the topic of link recovery. Although this paper does not address link recovery, these are important to consider when developing a traceability system. Traceability tools such as ADAMS [38], Poirot [39], ACTS [40], traceMaintainer [3] and Traceclipse [41] have been developed for link recovery and management in academia. DeLucia et al. [42] conduct a controlled experiment to assess the usefulness of an information retrieval based traceability recovery tool. Results show that the tool reduces the time spent vs. manual tracing. These studies compare different traceability techniques instead of assessing the effect of traceability itself on maintenance tasks. None of the studies look at the effect of traceability links on software maintenance tasks: the main goal of the study we present in this paper. A secondary goal of this paper was to gather data on what features are needed in a traceability system/tool that can be adopted successfully in industry.

There has also been some work on the visualization of traceability links. Marcus et al. [43] was the first paper attempting to visualize traceability links with TraceViz, and presents a list of visualization requirements. More recently Chen's work [44] focuses on visualizing traceability links in a project using a graph toolkit in Eclipse. VisMatrix [45] generates a graphical representation of the requirements traceability matrix. None of the above work conduct empirical studies to determine the feasibility of these visual representations in practice. We consider this paper to be the first attempt to conduct an empirical study to understand what industry developers actually need in a traceability tool. It takes a similar but small scale empirical approach as Ramesh et al. [6], by conducting a survey to determine what types of links practitioners are interested in. The work presented in this paper addresses the types of links sought by stakeholders (via an industrial survey) for collaboration. This is a first attempt at determining the types of links that are most useful to each different stakeholder. In addition, a study conducted on software maintenance tasks was used to determine the benefits of a traceability links for artifact-based collaboration in practice.

Gotel and Finkelstein [46] were one of the first researchers to do an empirical study among software developers in a large industrial firm. Their main goal was to understand the scale of traceability problem. They did not consider the effect of traceability links on maintenance tasks. Ramesh et al. [6] also conduct a study on industry practitioners by gathering data via questionnaires to better understand traceability links used. They define reference models based on their survey. Similar studies [47], [48] like the ones by Gotel and Ramesh have been done on a smaller scale but none of them look into the effects of traceability links on software maintenance tasks versus not having the links. Our study seeks to provide via empirical evidence that when developers are given traceability links to work with, they are better at software maintenance tasks.

Asuncion et al. [49] present an end-to-end traceability tool developed at a software firm that supported the entire life-cycle and focused on requirements traceability and the traceability process. Several guidelines were presented after deploying the traceability tool within the company. The above work complements the work in this paper. Neumuller et al. [50] also developed and applied a traceability system in a small company and reported their findings. The findings from our study corroborate with [50]; in both cases a need for better tools is called for. The study presented in this paper is a controlled experiment and differs from [50] because in this study, focused software maintenance tasks were used to determine the usefulness of the links via a prototype tool, *TraceLink*. Heindl et al. [51] discussed the application of requirements traceability to risk assessment and developed a cost-benefit model to help project managers. Klimpke et al. [2] conducted a case study at five industrial firms to determine how traceability was realized in practice. They also pointed out what needed to be considered in order to adopt traceability in industry. They found that companies used traceability in an ad hoc manner since there was a lack of existing tools that were customizable for their needs. Even though the study presented in this paper was done at only one firm, the findings were similar to those of Klimpke et al.

There have been several recent papers related to improving the performance of requirements to code traceability.

We describe some of them next. Yu et al. [52] present an invariant traceability framework to merge changes that occur between user-modified code and template-generated code in model-driven development. They focus on maintaining the bidirectional traceability links as software changes; our paper is concerned with showing the benefit of having the links while developers perform maintenance tasks. The goal in this paper was not to create a traceability tool but to provide evidence that traceability links actually do help in software maintenance tasks. Charrada et al. [53] present an approach to automatically detect outdated requirements when source code changes. The links are updated when the system evolves over time. This is important because a traceability system is only as good as the validity of its links to requirements. Mahmoud et al. [54] use refactorings to improve the performance of requirements-to-code traceability. They showed that as a system evolves, when corrupted textual and lexical structure was restored with refactorings, a positive impact on traceability was reported. Niu et al. [55] and Dekhtyar et al. [56] study the human analyst's behavior in automated tracing. They note that the quality of the requirements traceability matrix is important. They examine both the traceability matrix quality and the rational decision making process of human analysts while they determine validity of links.

This papers mentioned in the above paragraph aim at supporting the evolution of traceability links and/or requirements as software systems evolve or understanding how analysts validate traceability links. They do not show evidence that these links are actually useful to developers in software maintenance tasks. All previous evidence was purely anecdotal. Since our goal was to determine the effect of link presence, we considered the actual tool used to be of secondary importance in this paper. For this reason, we created a simple prototype that provided us with enough linking to run the study. Any tool that provides traceability link information would suffice since the tool itself is not being evaluated, but the presence of traceability links is.

## IX. CONCLUSION

This paper presents a study to assess the effect of the presence of traceability links during software maintenance tasks. A prototype tool namely *TraceLink* was developed to test the hypotheses in the study. Twenty-eight subjects from industry and academia participated in the study. Results indicate a significant increase in task accuracy when traceability links were used to solve the tasks. Subjects using traceability links were 86.06% more accurate. Using traceability links (via *TraceLink*) also reduced the gap between high and low ability subjects as well as between academic and industry subjects. This implies that using a traceability system provides more gains for low ability subjects making them more productive. Difficult tasks benefitted more from traceability links than the ones that were easier to solve. This is all the more evidence to adopt traceability practices in organizations.

In addition, an industrial survey was conducted before the study to determine the kinds of traceability links most needed by industry stakeholders for collaboration. Results from the survey include a traceability meta-model highlighting the strong relationship between the types of links needed and the stakeholder's role of developer, tester, and project manager. This is another important contribution of this paper. Several real-life scenarios are also discussed due to the lack of traceability at the industrial firm surveyed.

In future work, we plan to study traceability systems and interested links to different stakeholders at other industrial firms, as well as conduct a study that evaluates the benefits of a traceability system that provide views (defined in Section III), specific to stakeholders' roles. Furthermore, a reevaluation of the $H_2$ hypothesis (in Section IV), needs to occur with more subjects and more in-depth training on the traceability tool. These replications will strengthen the findings presented here.

## ACKNOWLEDGMENT

The authors thank all students and practitioners for their participation in the survey and the study.

## REFERENCES

[1] C. Treude, M. Storey, and J. Weber, *Empirical Studies on Collaboration in Software Development: A Systematic Literature Review*. Victoria, BC, Canada: University of Victoria, 2009.

[2] L. Klimpke and T. Hildenbrand, "Towards end-to-end traceability: Insights and implications from five case studies," in *Proc. Int. Conf. Softw. Eng. Adv.*, Porto, Portugal, 2009, pp. 465–470.

[3] P. Mader, O. Gotel, and I. Philippow, "Motivation matters in the traceability trenches," in *Proc. 17th IEEE Int. RE Conf.*, Aug./Sep. 2009, pp. 143–148.

[4] G. Spanoudakis and A. Zisman, "Software traceability: A roadmap," in *Handbook of Software Engineering and Knowledge Engineering*, S. K. Chang, Ed. Singapore: World Scientific, 2005, pp. 395–428.

[5] G. Antoniol, B. Berenbach, A. Egyed, S. Ferguson, J. I. Maletic, and A. Zisman, *Problem Statements and Grand Challenges, Center of Excellence for Traceability*. Lexington, KY, USA: Center Excellence Softw. Traceability, 2006.

[6] B. Ramesh and M. Jarke, "Towards reference models for requirements traceability," *IEEE Trans. Softw. Eng.*, vol. 27, no. 1, pp. 58–93, Jan. 2001.

[7] P. Mader, O. Gotel, and I. Philippow, "Getting back to basics: Promoting the use of a traceability information model in practice," in *Proc. ICSE Workshop TEFSE*, May 2009, pp. 21–25.

[8] J. I. Maletic, M. L. Collard, and B. Simoes, "An XML-based approach to support the evolution of model-to-model traceability links," in *Proc. 3rd ACM Int. Workshop TEFSE*, Long Beach, CA, USA, 2005, pp. 67–72.

[9] J. I. Maletic, E. V. Munson, A. Marcus, and T. N. Nguyen, "Using a hypertext model for traceability link conformance analysis," in *Proc. 2nd Int. Workshop TEFSE*, Montreal, QC, Canada, 2003, pp. 47–54.

[10] P. Spirtes, C. Glymour, and R. Scheines, *Causation, Prediction, and Search*. Cambridge, MA, USA: MIT Press, 2001.

[11] G. Spanoudakis, A. Zisman, E. Perez-Minana, and P. Krause, "Rule-based generation of requirements traceability relations," *J. Syst. Softw.*, vol. 72, no. 2, pp. 105–127, 2004.

[12] L. C. Lamb, W. Jirapanthong, and A. Zisman, "Formalizing traceability relations for product lines," in *Proc. 6th Int. Workshop TEFSE*, Honolulu, HI, USA, 2011, pp. 42–45.

[13] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering—An Introduction*. Norwell, MA, USA: Kluwer, 1999.

[14] A. Meneely, B. Smith, and L. Williams, *iTrust Electronic Health Care System: A Case Study*. New York, NY, USA: Springer-Verlag, 2012.

[15] T. Hildenbrand, F. Rothlauf, M. Geisser, A. Heinzl, and T. Kude, "Approaches to collaborative software development," in *Proc. 2nd Int. Conf. Complex Intell. Softw. Intensive Syst.*, Mar. 2008, pp. 523–528.

[16] J. Whitehead, "Collaboration in software engineering: A roadmap," in *Proc. FOSE*, 2007, pp. 214–225.
[17] L. Pilatti, J. L. N. Audy, and R. Prikladnicki, "Software configuration management over a global software development environment: Lessons learned from a case study," in *Proc. Int. Workshop Global Softw. Develop. Practitioner*, New York, NY, USA, 2006, pp. 45–50.
[18] J. Brown, G. Lindgaard, and R. Biddle, "Stories, sketches, and lists: Developers and interaction designers interacting through artefacts," in *Proc. AGILE*, 2008, pp. 39–50.
[19] R. E. Grinter, "Using a configuration management tool to coordinate software development," in *Proc. Org. Comput. Syst.*, New York, NY, USA, 1995, pp. 168–177.
[20] M. Handel and J. D. Herbsleb, "What is chat doing in the workplace?" in *Proc. ACM Conf. CSCW*, New York, NY, USA, 2002, pp. 1–10.
[21] E. Isaacs, A. Walendowski, S. Whittaker, D. J. Schiano, and C. Kamm, "The character, functions, and styles of instant messaging in the workplace," in *Proc. ACM Conf. Comput. Supported Cooperat. Work*, New York, NY, USA, 2002, pp. 11–20.
[22] J. J. Cadiz, A. Gupta, and J. Grudin, "Using web annotations for asynchronous collaboration around documents," in *Proc. Comput. Supported Cooperat. Work*, New York, NY, USA, 2000, pp. 309–318.
[23] M.-A. Storey, J. Ryall, R. I. Bull, D. Myers, and J. Singer, "TODO or to bug: Exploring how task annotations play a role in the work practices of software developers," in *Proc. 30th Int. Conf. Softw. Eng.*, New York, NY, USA, 2008, pp. 251–260.
[24] A. Pandey, C. Miklos, M. Paul, N. Kameli, F. Boudigou, V. Vijay, *et al.*, Application of tightly coupled engineering team for development of test automation software—A real world experience," in *Proc. Comput. Softw. Appl. Conf.*, 2003, pp. 56–63.
[25] A. Sarma, D. Redmiles, and A. van der Hoek, "Empirical evidence of the benefits of workspace awareness in software configuration management," in *Proc. 16th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, New York, NY, USA, 2008, pp. 113–123.
[26] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson, "FASTdash: A visual dashboard for fostering awareness in software teams," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, New York, NY, USA, 2007, pp. 1313–1322.
[27] S. Zhang, C. Zhao, Q. Zhang, H. Su, H. Guo, J. Cui, *et al.*, "Managing collaborative activities in project management," in *Proc. Symp. Comput. Human Int. Manag. Inf. Technol.*, New York, NY, USA, 2007, article no. 3.
[28] B. Tessem and J. Iden, "Cooperation between developers and operations in software engineering projects," in *Proc. Int. Workshop CHASE*, New York, NY, USA, 2008, pp. 105–103.
[29] B. Sharif and J. I. Maletic, "Using fine-grained differencing to evolve traceability links," in *Proc. ACM Int. Symp. Grand Challenges Traceabil.*, Lexington, KY, USA, 2007, pp. 76–81.
[30] P. Mäder and A. Egyed, "Assessing the effect of requirements traceability for software maintenance," in *Proc. ICSM*, 2012, pp. 171–180.
[31] A. De Lucia, F. Fasano, R. Oliveto, and M. Via Ponte don, "Traceability management for impact analysis," in *Proc. Frontiers Softw. Maintenance*, Beijing, China, 2008, pp. 21–30.
[32] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *Proc. 25th IEEE/ACM ICSE*, Portland, OR, USA, 2003, pp. 125–137.
[33] H. Kagdi, J. I. Maletic, and B. Sharif, "Mining software repositories for traceability links," in *Proc. IEEE ICPC*, Jun. 2007, pp. 145–154.
[34] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artefact management systems using information retrieval methods," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 4, pp. 13:1–13:50, 2007.
[35] C. McMillan, D. Poshyvanyk, and M. Revelle, "Combining textual and structural analysis of software artifacts for traceability link recovery," in *Proc. ICSE Workshop Traceabil. Emerg. Forms Softw. Eng.*, 2009, pp. 41–48.
[36] X. Zou, R. Settimi, and J. Cleland-Huang, "Improving automated requirements trace retrieval: A study of term-based enhancement methods, *Empirical Softw. Eng.*, vol. 15, no. 2, pp. 119–146, 2010.
[37] K. Sundaram, J. H. Hayes, A. Dekhtyar, and A. E. Holbrook, "Assessing traceability of software engineering artifacts," *Requirements Eng. J.*, vol. 15, no. 3, pp. 313–335, 2010.
[38] A. DeLucia, F. Fasano, R. Oliveto, and G. Tortora, "ADAMS: Advanced artefact management system," in *Proc. 10th Eur. Conf. Softw. Maintenance Reeng.*, Bari, Italy, 2006, pp. 349–350.
[39] J. Lin, C. C. Lin, J. Cleland-Huang, R. Settimi, J. Amaya, G. Bedford, *et al.*, "Poirot: A distributed tool supporting enterprise-wide traceability," in *Proc. 14th IEEE Int. Conf. RE*, Sep. 2006, pp. 363–364.
[40] H. Asuncion, A. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng.*, Cape Town, South Africa, 2010, pp. 95–104.
[41] S. Klock, M. Gethers, B. Dit, and D. Poshyvanyk, "Traceclipse: An eclipse plug-in for traceability link recovery and management," in *Proc. 6th Int. Workshop Traceabil. Emerg. Forms Software Eng.*, Honolulu, HI, USA, 2011, pp. 24–30.
[42] A. De Lucia, R. Oliveto, and G. Tortora, "Assessing IR-based traceability recovery tools through controlled experiments," *Empirical Softw. Eng.*, vol. 14, no. 1, pp. 57–92, 2009.
[43] A. Marcus, X. Xie, and D. Poshyvanyk, "When and how to visualize traceability links?" in *Proc. 3rd ACM Int. Workshop Traceabil. Emerg. Forms Softw. Eng.*, Long Beach, CA, USA, 2005, pp. 56–61.
[44] X. Chen, "Extraction and visualization of traceability relationships between documents and source code," in *Proc. IEEE/ACM Int. Conf. Autom. Softw. Eng.*, Antwerp, Belgium, 2010, pp. 505–510.
[45] C. Duan and J. Cleland-Huang, "Visualization and analysis in automated trace retrieval," in *Proc. 1st Int. Workshop Requirem. Eng. Visualizat.*, Minneapolis, MN, USA, Sep. 2006, p. 5, doi: 10.1109/REV.2006.6.
[46] O. C. Z. Gotel and A. C. W. Finkelstein, "An analysis of the requirements traceability problem," in *Proc. 1st IEEE Int. Conf. Requirm. Eng.*, Apr. 1994, pp. 94–101.
[47] P. Arkley and S. Riddle, "Overcoming the traceability benefit problem," in *Proc. 13th IEEE Int. Conf. RE*, Paris, France, Aug./Sep. 2005, pp. 385–389.
[48] A. Ahmad and M. A. Ghazali, "Documenting requirements traceability information for small projects," in *Proc. IEEE Int. Multitopic Conf.*, Dec. 2007, pp. 1–5.
[49] H. Asuncion, F. Francois, and R. N. Taylor, "An end-to-end industrial software traceability tool," in *Proc. 6th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, Dubrovnik, Croatia, 2007, pp. 115–124.
[50] C. Neumuller and P. Grunbacher, "Automating software traceability in very small companies: A case study and lessons learned," in *Proc. 21st IEEE Int. Conf. Autom. Softw. Eng.*, Tokyo, Japan, 2006, pp. 145–156.
[51] M. Heindl and S. Biffl, "Risk management with enhanced tracing of requirements rationale in highly distributed projects," in *Proc. Int. Workshop Global Softw. Develop. Practitioner*, Shanghai, China, 2006, pp. 20–26.
[52] Y. Yu, Y. Lin, Z. Hu, S. Hidaka, H. Kato, and L. Montrieux, "Maintaining invariant traceability through bidirectional transformations," in *Proc. 34th ICSE*, Zurich, Switzerland, 2012, pp. 540–550.
[53] E. Charrada, A. Koziolek, and M. Glinz, "Identifying outdated requirements based on source code changes," in *Proc. 20th IEEE Int. RE Conf.*, Chicago, IL, USA, Sep. 2012, pp. 61–70.
[54] A. Mahmoud and N. Niu, "Supporting requirements traceability through refactoring," in *Proc. 21st Int. RE Conf.*, 2013, pp. 32–41.
[55] N. Niu, A. Mahmoud, Z. Chen, and G. Bradshaw, "Departures from optimality: Understanding human analyst's information foraging in assisted requirements tracing," in *Proc. ICSE*, San Francisco, CA, USA, 2013, pp. 572–581.
[56] 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 *Proc. 19th IEEE Int. RE Conf.*, Aug./Sep. 2011, pp. 111–120.

**KHALED JABER** is currently pursuing the Ph.D. degree with the Electrical and Engineering and Computer Science, Ohio University, Athens, OH, USA. He received the B.S. degree in mathematics from An-Najah University, Palestine, in 1985, and the M.S. degree in computer science from Northeastern Illinois University, Chicago, IL, USA, in 1991. He was with Lucent Technologies and Emerson developing and managing software systems. His research interest includes software engineering, 3-D visualization, and project management.

**BONITA SHARIF** is an Assistant Professor with the Department of Computer Science and Information Systems, Youngstown State University, OH, USA. Her research interests are in empirical software engineering, software traceability, software visualization to support maintenance of large systems, and eye tracking research related to software engineering. She received the Ph.D. and M.S. degrees in computer science from Kent State University, USA, in 2010 and 2003, respectively, and the B.S. degree in computer science from Cyprus College, Nicosia Cyprus. Prior to this position, she was an Adjunct Assistant Professor at Ohio University.

**CHANG LIU** (M'96–SM'12) received the B.S. degree in computer science from Fudan University, Shanghai, China, in 1991, and the Ph.D. degree in information and computer science from the University of California, Irvine, in 2002. He is currently an Associate Professor of electrical engineering and computer science with Ohio University. He has published over 30 refereed papers and won over 20 grants totaling over five million dollars. Sponsors of his projects include the National Science Foundation, the United States Environmental Protection Agency, and the Ohio Environmental Protection Agency. His research interests include software engineering, 3-D immersive learning games, and mobile health applications. He is a Lifetime Senior Member of the ACM. He was a recipient of the 2009 Marvin E. and Ann D. White Research Award and the 2007 Advanced Technology Summit Award for Leadership.