International Journal of Software Engineering and Knowledge Engineering
Vol. 0, No. 0 (1994) 000–000

A FRAMEWORK FOR COMPARING REQUIREMENTS TRACING EXPERIMENTS

JANE HUFFMAN HAYES
Department of Computer Science, University of Kentucky, 773 Anderson Hall
Lexington, KY, 40506, USA Country

and

ALEX DEKHTYAR
Department of Computer Science, University of Kentucky, 773 Anderson Hall
Lexington, KY, 40506, USA

Received (received date)
Revised (revised date)
Accepted (accepted date)

The building of traceability matrices by those other than the original developers is an arduous, error prone, prolonged, and labor intensive task. Thus, after the fact requirements tracing is a process where the right kind of automation can definitely assist an analyst. Recently, a number of researchers have studied the application of various methods, often based on information retrieval, to after the fact tracing. The studies are diverse enough to warrant a means for comparing them easily as well as for determining areas that require further investigation. To that end, we present here an experimental framework for evaluating requirements tracing and traceability studies. Common methods, metrics and measures are described. Recent experimental requirements tracing journal and conference papers are catalogued using the framework. We compare these studies and identify areas for future research. Finally, we provide suggestions for how the field of tracing and traceability research may move to a more mature level.

Keywords: requirements tracing, traceability, experiment, framework, metrics, information retrieval, IV& V, case study

1. Introduction

Requirements tracing is defined as "the ability to describe and follow the life of a requirement, in both a forward and backward direction, through the whole systems life cycle [30]." Requirements tracing is important to the software engineering field for a number of reasons: traceability matrices (a) assist us in assuring that all requirements have been implemented, (b) participate in mapping test cases to requirements, (c) are used by management for "what if" scenarios, (d) assist us when we maintain software or reuse software, and (e) form a part of the safety case for safety-critical software requiring certification. Unfortunately, traceability matrices are often not built, or not to the level of detail required, during the development effort. As a result, they must be built after the fact by non-developers. Tools and techniques to assist with the automation of this time consuming, highly error prone, unpleasant task are needed.

On a positive note, in the last three to five years, there have been an increased number of research papers in the area of requirements tracing. Specifically, many of these papers [3, 29, 24, 23] apply information retrieval (IR) methods to the requirements tracing problem in a variety of settings. In particular, Information Retrieval methods are used to compare the texts of a pair of requirements from two documents in the project document hierarchy for the purpose of determining their similarity. This work uses well-accepted IR measures of *recall* and *precision* to evaluate the effectiveness of their techniques. Recall is the percentage of actual matches that are found and precision is the percentage of correct matches as a ratio to the total number of candidate links returned [23]. Beyond the use of these two measures, the papers in this field have little in common. Some papers introduce secondary measures of effectiveness or quality [24]. Some papers take an experimental approach to their evaluation, others take a less formal approach. As more and more traceability studies become available, the need for a clearly outlined experimental framework that allows side-by-side comparison is emerging.

Fenton, Pfleeger, and Glass [17] point out that far too often software engineering researchers rely on intuition and not empirical research and data. Basili, Shull, and Lanubile [8] examine a number of papers related to software engineering experimental frameworks. They note: "the important common characteristic of all these frameworks is that they document the key choices made during experimental design, along with their rationales." Further, the frameworks allow the comparison of studies and "allow the primary question of an experiment to shift from 'Is a particular process effective?' to 'What are the factors that make a particular process effective or ineffective?' [8]."

The need for such a framework in software engineering experimentation has been acknowledged. Similarly, the importance of applying such a framework to experiments of defect-detection techniques has also been demonstrated [28]. There is a need for a framework or structure for empirical studies on requirements tracing. To that end, this paper discusses experiments that examine techniques for tracing artifacts such as requirements. The main contribution of this paper is two-fold:

Framework for requirements tracing experiments. We propose a framework for developing, conducting and analyzing experiments on requirements traceability. This framework allows description of various existing and emerging research on requirements tracing and traceability.

Description of existing research in terms of the proposed framework. We provide descriptions of [3, 29, 4, 23] in terms of our framework. Such descriptions allow us to compare two or more works side-by-side, and determine which areas of requirements traceability research have not been adequately addressed, and design future experiments to cover them.

The rest of the paper is organized as follows. Section 2 presents related work in the area of requirements tracing, traceability and experimentation. In Section 3 we briefly describe our model of the requirements tracing process and how IR methods are applied to it. A framework for the many factors to be considered in requirements tracing experiments and proposed requirements tracing measures is presented in Section 4. Section 5 examines recent experiments that evaluate requirements tracing techniques. These experiments provide hypotheses, experimental designs, and sometimes objects of experimentation (such as techniques, models, project artifacts) that can be reused. We emphasize an experiment that we conducted at the University of Kentucky. Finally, conclusions and future work are discussed in Section 6.

2. Related Work

2.1. *Requirements tracing*

We have been tackling the requirements tracing problem for many decades. In 1978, Pierce [32] designed a requirements tracing tool as a way to build and maintain a requirements database and facilitate requirements analysis and system verification and validation for a large Navy undersea acoustic sensor system. Hayes [21] discusses a front end for a requirements tracing tool called the Software Automated Verification and Validation and Analysis System (SAVVAS) Front End processor (SFEP). This was written in Pascal and interfaced with the SAVVAS requirements tracing tool that was based on an Ingres relational database. SFEP allows the extraction of requirement text as well as the assignment of requirement keywords through the use of specified linkwords such as "shall", "must", "will", etc. These tools are largely based on keyword matching and threshold setting for that matching. Several years later, the tools were ported to hypercard technology on Macs, and then to Microsoft Access and Visual Basic running on PCs. This work is described by Mundie and Hallsworth in [31]. These tools have since been further enhanced and are still in use as part of the Independent Verification and Validation (IV&V) efforts for the Mission Planning system of the Tomahawk Cruise Missile as well as for several NASA Code S science projects.

Abrahams and Barkley, Ramesh, and Watkins and Neal [1, 33, 39] discuss the importance of requirements tracing from a developer's perspective and explain basic concepts such as forward, backward, vertical, and horizontal tracing. Casotto [13] examined run-time tracing of the design activity. Her approach uses requirement cards organized into linear hierarchical stacks and supports retracing. Tsumaki and Morisawa [38] discuss requirements tracing using UML. Specifically they look at tracing artifacts such as use-cases, class diagrams, and sequence diagrams from the business model to the analysis model and to the design model (and back) [38].

There have also been significant advances in the area of requirements elicitation, analysis, and tracing. Work has been based on lexical analysis, such as extraction and analysis of phoneme occurrences to categorize and analyze requirements and other artifacts [34]. Bohner's work on software change impact analysis using a graphing technique may be useful in performing tracing of changed requirements [10]. Anezin and Brouse advance backward tracing and multimedia requirements tracing in [2, 12].

Gotel and Finkelstein [18] examined the usefulness of contribution structures in a 3-year long industrial study. Contribution structures allow personnel-related traceability, focusing on the human sources of requirements. Their study found that contribution structures "identified the right people to help rectify matters where problems of misunderstanding surfaced, to consider requirements change and to handle staff turn-over." Social roles and relations were also more easily discerned. Zisman et al. [41] define traceability relations based on requirements-to-object-model and inter-requirements traceability rules. Their prototype tool allows the generation of relations between commercial requirements specifications (features for a family of products - specified in natural language) and functional requirements specifications (behavior for a family of products - specified in use cases) and the requirements object model (UML). A case study of 110 classes and 277 operations showed that recall of 76% and precision of 31% to 100% can be achieved, depending on the rule examined. This work was continued in Spanoudakis et al [37]. The presented technique generates traceability relations by using a rule-base. Some advancements reported include: ability to determine the type of links found ("requires execution of," "requires feature in"), and the ability to handle sections of textual use cases and analysis models (defined in XML). After building a prototype and running some preliminary experiments, they were able to achieve recall of up to 95

Cleland-Huang et al. [14] propose an event-based traceability technique for supporting impact analysis of performance requirements. Data is propagated speculatively into performance models that are then re-executed to determine impacts from the proposed change. Ramesh et al examine reference models for traceability. They establish two specific models, a low-end model of traceability and a high-end model of traceability for more sophisticated users [33]. They found that a typical low end user created traceability links to model requirement dependencies, to examine how requirements had been allocated to system components, to verify that requirements had been satisfied, and to assist with change control. A typical high-end user, on the other hand, uses traceability for full coverage of the life cycle, includes the user and the customer in this process, captures discussion issues, decision, and rationale, and captures traces across product and process dimensions [33].

2.2. *Information Retrieval in Requirements Analysis*

In general, the software tools described above address the overall problem of requirements management during the lifecycle of a software project. Their requirements tracing components typically rely, one way or another, on manual keyword assignment - a long and arduous process. With time, practitioners realized the potential benefits of, and the researchers started working on, methods for automating the requirements tracing process. Of the many methods examined, Information Retrieval techniques appear to offer much promise for this automation.

Two research groups worked on requirements-to-code tracebility. Antoniol, Canfora, De Lucia and Merlo [3] considered two IR methods: probabilistic IR and vector retrieval (tf-idf). They have studied the traceability of requirements to code for two datasets. In their testing, they retrieved the top i matches for each requirement for $i = 1, 2, \ldots$ and computed precision and recall for each i. Using improved processes, they were able to achieve 100% recall at 13.8% precision for one of the datasets. In general, they have achieved encouraging results for both tf-idf and probabilistic IR methods. Following [3], Marcus and Maletic [29] applied the latent semantic indexing (LSI) technique to the same problem. In their work they used the same datasets and the same retrieval tests as [3]. They have shown that LSI methods show consistent improvement in precision and recall and were able to achieve combinations of 93.5% recall and 54% precision for one of the datasets.

While [3] and [29] studied requirements-to-code traceability, in [23] we have addressed the problem of tracing requirements between different documents in the project document hierarchy. In the preliminary study [23], we have implemented three methods: tf-idf, tf-idf with key phrases and tf-idf with simple thesaurus. We reported on their success in identifying links between two requirements documents. In our study, retrieval with simple thesaurus outperformed other methods on our test dataset, producing recall of 85% with precision of 40%. [24] continues the research started in [23]. We extended the baseline tf-idf and thesaurus retrieval methods with analyst relevance feedback processing capability [24].

While [23] concentrated solely on the problem of candidate link generation, [24] looked at the entire process of requirements tracing from the perspective of the performing analyst. There, we proposed a number of non-functional requirements for software tools designed to assist analysts in tracing requirements, determined means for evaluating these requirements, and described a study that showed that our prototype tool RETRO (REquirements TRacing On-target) matches the objective components of the proposed requirements.* The key difference between the experimental designs of [23] and [24] was that in the latter paper we used *feedback processing* techniques to emulate analyst interaction with the tool and looked at the improvement in the metrics over the iterations of the process. To better understand the structural changes in the links of candidate lists returned by RETRO at different iterations of the process, we have introduced a number of *secondary metrics* that measure *separation* between the true links and the false-positives in those lists. Our experiments have shown that together with the improvement in the primary metrics (recall and precision), we are achieving better separation, i.e., true links "rise" to the top of candidate link lists while false-positives "sink" to the bottom.

*The requirements proposed in [24] have two components: objective, that can be evaluated by studying the software outputs, and subjective, that can only be evaluated by studying the work of human analysts with the tool and their reactions to the outputs. The latter study is currently in development stages.

2.2.1. *Information retrieval measures*

Information Retrieval uses two key traditional measures: *precision* and *recall* (see, for example, [6]) to evaluate the performance of different methods. These measures are discussed in more detail in Section 3.2. These measures are applicable to the treceability analysis in general — regardless of whether a specific tracing task is performed using an automated method or manually: precision tells us the percentage of correct links in the final list of links while recall specifies the overall percentage of correct links discovered.

The scope of this paper is broader than the study of individual measures of performance of tracing methods. Rather, we discuss an overall framework for conducting tracing studies. This framework allows us to compare different tracing experiments not only based on these quantitative measures, but also on a broad range of qualitative features, from the purpose and size of the study, to the study conclusions. We should note, however, that the quantitative measures of performance do play an important role in the proposed framework. Precision and recall are accepted as the key measures in the Information Retrieval community. In [25] we have studied the question of applicability of precision and recall to tracing experiments in detail, and discussed a number of *secondary measures*, that improve the understanding of the results of tracing experiments. As such, in this paper, we do not concentrate on specific ways to measure the performance of tracing experiments. Our case studies [3, 29, 23], and other related research [37, 41], use precision and recall. In [24, 25], we show some new measures, *selectivity*, *lag* and *DiffAR*, to be useful when analyzing and comparing tracing experiments.

2.3. *Experimentation*

In 1986, Basili, Selby, and Hutchens described a framework that allowed the categorization, description, and understanding of software engineering research experiments [7]. According to Bourque and Abran [11], this framework was never used by Basili et al or other researchers, hence they called for a field test of the framework. As a result of a researcher's workshop on empirical studies, Lanubile [27] proposed a framework similar to that of [7], but provided different attributes for each of the classification dimensions. For example, focus of the study could be on a single, specific object of study or on multiple, generic objects [27, 8]. Lott and Rombach [28] present a framework for repeating and comparing software engineering experiments. Their characterization scheme was specifically designed to compare defect-detection techniques, but can be used in a more general way also. Their framework adds detail in the area of the Experimental Plan. For example, under data collection and validation procedures, researchers must specify how online and off-line collection procedures were used (forms, videotapes, counts of runs) as well as validation approaches (independent sources, interviews, etc.) [28]. This paper is organized similarly to Lott and Rombach [28]. Fenton, Pfleeger, and Glass examine five questions related to software engineering research and experimentation in [17]. Specifically, they ask researchers to examine whether their work is based on empirical research and data, whether the experiment is designed properly, whether toy situations are studied, whether appropriate measures are used, and whether or not the study is conducted for a long enough period of time. Hayes applied the Basili et al framework [7] to real-world projects that also double as experimental studies [22].

3. Information Retrieval for Requirements Tracing

In this section, we briefly describe the requirements tracing process from the point of view of the performing analyst. While the experimental framework that we describe in Section 4 is independent of the specific methodology that is used in the requirements tracing process, Section 5 applies the framework to research that used IR methods. Thus, for the sake of completeness, we include a brief outline of how IR methods are used to support the requirements tracing process and give a short survey of specific techniques used in [3, 29, 24, 23].

3.1. *The Process of Requirements Tracing*

Ideally, the requirements traceability matrix for any pair of documents within the project document hierarchy should be a by-product of the development effort. That is, any time developers work on a lower level requirements document based on a higher level document, the traceability information should be generated and inserted in the document at the time of introduction of individual lower level requirements.

In practice, however, very few development teams follow this approach. Thus, requirements tracing becomes a part of the Independent Validation and Verification (IV& V) or V& V process, performed by analysts who were *not* part of the original development team (and often work for a different company).

In their work on specific requirements tracing tasks, IV& V analysts rely solely on the project artifacts provided to them by the development team. First and formemost, these are the actual requirements documents, typically a higher level document that needs to be traced and a lower level document to which it needs to be traced. IV & V analysts may also use other artifacts, both textual (such as project dictionaries and code) and non-textual (such as UML diagrams, use case diagrams, etc.).

In a nutshell, the requirements tracing process can be described as follows (we describe the process for a trace of a high level to a low level document, but tracing can be applied to peer artifacts also). The analyst needs to generate a list of candidate links for each high level requirement. This list includes low level requirements which should be examined closely to determine if they satisfy, at least in part, the high level requirement in question. For each pair of high and low level requirements from such a list, the analyst must then make a binary "link" / "no link" decision. The process thus gets separated into two major stages: (a) generation of the candidate link lists and (b) evaluation of the candidate links from the generated lists.

Generation of candidate link lists. Prior to the use of IR methods for candidate link generation, the main techniques included (i) manual study of the documents in hardcopy, (ii) manual study of the documents in softcopy, (iii) use of traditional office software (text editors, spreadsheets), and (iv) use of special purpose requirements management software. The most naive approach in each case is to consider all possible pairs of high and low level requirements. This is very costly, however. Given that, typically, the number of matching low level requirements per a high level requirement is *much smaller* than the total number of low level requirements, it is also quite wasteful. Traditional methods used to avoid exhaustive search consist of (a) assignment of keywords to each individual high and low level requirement and (b) insertion into the list of candidate links all pairs of high-low requirements that have at least one common assigned keyword.

In all four possible procedures mentioned above (i through iv), the assignment of keywords to requirements is a *manual process*. At the same time, keyword matching is done manually in the first two; is performed using text editor search facilities in the third; and, typically, is performed completely automatically in the fourth.

Evaluation of candiate links. Each pair of requirements deemed "suspicious" during candidate link list generation needs to be examined more closely. Upon this examination, the analyst pronounces his/her final "link" or "no link" verdict. We note, that in order for the results of the IV& V inspection to be trustworthy, this part of the process **must always be performed manually**. Requirements management software might make this process more convenient for the user by providing a comfortable, informative interface. In the end, though, it is the human judgment that is used to make the final determination.

The actual process of examination of a given candidate link differs from analyst to analyst, project to project, and candidate link to candidate link. Generally speaking, the analyst studies the text of both requirements as well as any accompanying non-textual components, determines respective positions of high and low level requirements in the document, and makes the judgment call on whether the low level requirement had been (purposefully or indavertantly) written to satisfy the high level requirement. As mentioned above, the analyst may choose to consult some additional projct artifacts before arriving at this judgment.

3.2. *Enter Information Retrieval*

We observe that in the above, somewhat simplified, description of the "pre-IR" requirements tracing process, the bottleneck lies in generation of the candidate links list. As stated above, the final judgement about the appropriateness of each considered candidate link must remain with the human analyst. Therefore, the total time spent on requirements tracing is in direct proportion to the total number of candidate links in the generated list.

While keyword-matching support provided by requirements management software, such as SuperTracePlus [21, 31], results in significant improvement of the process, it still leaves the initial examination of requirements and keyword assignment to the analysts. Among the drawbacks of such a process is the proneness to typical human errors, such as inconsistent assignement of keywords (e.g., "fault" in one place, "error" in another), missed keywords (due to lack of attention and/or simple tiredness), and lapses in judgment (such as incorrect choice of keywords due to misunderstanding of the meaning of the requirement).

Information Retrieval methods, battle-tested in the past 20–25 years and popularized by the emergence of web search engines as the keystones to world wide web surfing, provide reliable and scalable mechanisms for keyword-matching between different documents in their simple form. In the general context of requirements tracing, individual requirements take on the roles of IR "documents" and "queries" or "information needs." In the standard setting of forward tracing (from a high level document to a lower level document), low level requirements become the documents or document collection, while high level requirements take on the role of queries.

The key advantage of IR methods over manual keyword assignment is efficiency – IR algorithms automate selection of keywords, determination of their relative importance to each requirement, and computation of similarity/degree of match between the text of high and low level requirements. At the same time, while IR methods are not subject to typical human errors such as missed keywords, they are limited by their input – the text of the requirement. Unlike humans, IR methods cannot simply leap to judgment that the requirement `''The software shall not allow the user to enter incorrect dates"` might be associated with keyphrases "error handling" or "input processing," because these terms are not present in the text of the requirement.[†]

In the rest of this section we survey Information Retrieval methods and techniques which have been applied to the tracing problems in recent years as well as discuss in more detail the metrics used to evaluate the success of IR methods.

Vector Space Information Retrieval (VSIR) (used in [3, 23, 24]). One of the oldest, simplest, well-known, well-studied, and robust approaches to determining whether a specific document is relevant to a given query consists in (a) representing each document and each query as a *vector of keyword weights* and (b) computing the similarity between the vectors as the cosine of the angle between them in the N-dimensional space (where N is the total number of keywords found in the document collection) [6]. This method is also known as *tf-idf*, called so for the way by which the vectors of keyword weights are computed. *Tf* stands for "term frequency" – the (normalized) frequency of the term in a given document or query, while *idf* stands for *inverse document frequency*, computed as $idf(k_i) = \log(\frac{1}{n_i})$, where n_i is the number of documents in which keyword k_i occurs.[‡]

[†] In all fairness, more complex methods involving the use of term ontologies and thesauri may allow such conclusions, but still, such conclusions are pre-programmed by the data available to them.

[‡] Term frequency expresses the idea that the more frequent the word is in a document, the more important it is for the document, while inverse document frequency represents the discriminatory power of a word – words that occur in fewer documents distinguish between relevant and

Probabilistic Information Retrieval (PIR) (used in [3].) This method uses simplified vector representation of the documents and queries: each keyword weight is either equal to 1 (keyword is found in the document) or 0 (keyword is not found). The probabilistic IR method, also known as Binary Independence Retrieval (BIR)[35, 15], estimates the probability that document d is relevant to some query q given their binary vectors (representing keyword occurrence). We refer the reader to [15] for the complete derivation of the formulae used in this method.

Latent Semantic Indexing (LSI) (used in [29]). Latent semantic indexing technique, first proposed in [16], uses Single-Value Decomposition (SVD) of the document-by-keyword matrix (formed out of the tf-idf vectors of keyword weights) to reduce the number of dimensions over which the similarity computation is taking place. Formally, if A is an $M \times N$ document-by-term weight matrix, its SVD is written as $A = TSD'$, where T and D' are two matrices with orthogonal rows and columns respectively and S is a diagonal matrix of eigenvalues of A. By trimming the list of eigenvalues from $rank(A)$ to a smaller number k, we obtain an approximate decomposition $A_k = TS_kD'$, where S_k is the diagonal matrix of size $k \times k$ with k largest eigenvalues of A on the diagonal. For comparing documents to each other, and processing queries, we can now use matrix DS_k^2D' which reduces the dimensionality of the document vectors from N to k. In practical applications, LSI performed well, and showed its robustness. At the same time, the SVD process is quite time-consuming, resulting in LSI being a rather slow method, typically reserved for applications with reasonably small domains or applications where quality outweighs efficiency.

Use of Thesaurus (used in [23, 24]). Standard tf-idf method produces non-zero relevance weight iff at least one pair of keywords match in two documents. Because individual requirements are quite terse, and because requirements at different levels are written by different people, it is not uncommon for the texts of matching requirements not to have terms in common. For example, the high level requirement`‘‘the software shall correctly process incoming data in XML format’’` and low level requirement `‘‘run Apache parser on input file temp_info.xml. The DTD file is input.dtd (see Appendix)."` have no common terms. Yet, the low level requirement clearly links to the high level requirement.

To alleviate this problem, we enhanced tf-idf method with some simple *thesaurus* information [23, 24]. Our thesaurus is a set of triples (v, α, w), where v and w are terms or term phrases and α is the degree to which the two terms match each other. For example, to let the link between the two requirements above be discovered automatically, we can construct the following thesaurus entry: $(\texttt{incoming data}, 0.9, \texttt{input})$. Formulae used in constructing document and query vectors enhanced with simple thesaurus information, as well as in determining their similarity, can be found in [24].

nonrelevant documents better.

Metrics and measures. Two standard metrics used to evaluate IR methods in a complementary way, precision and recall, measure the accuracy of the answer set generated by an IR method on a given query. Let the size of the entire document collection be M, and let R documents be actually relevant to some query q. Suppose our IR method returns n documents, out of which r are the relevant documents. Then, precision of the experiment is defined as $precision = \frac{r}{m}$, while the recall is $recall = \frac{r}{R}$. These two metrics are used in all papers applying IR for requirements tracing.

In [24], we introduce some new measures designed to help us evaluate the quality of lists of candidate links generated by the iterative *feedback processor*. These measures are able to capture structural changes in the lists of candidate links even when the precision and recall do not change significantly. The two measures considered there were *Lag* - the mean number of false positives above a true link in the candidate link lists and *DiffAR* - the difference in the average relevance of true links and the average relevance of false positive links. These measures support our findings in [24] that the quality of the answer set keeps improving throughout the feedback iterations.

4. A Framework for Requirements Tracing Experiments

The contribution of the framework of this section is to help achieve the goal of an infrastructure for experimental software engineering experiments that evaluate requirements tracing techniques. The framework builds on work that appeared in [22]. It is depicted in Table 1. Some additions, modifications and/or changed interpretations have been made to tailor the framework to requirements tracing and traceability experiments. Hypothesis was added as a category after Lott and Rombach [28]. We replaced the selections under the experimental design category with a subset of those used by Lott and Rombach [28]. Importance has been divided into domain importance and object importance. We added a results category under the interpretation phase and we recognize two levels of results. The framework encompasses definition of the experimental study, planning of the study, realization of the study, and interpretation of the study, just as in [7].

4.1. *Definition*

Definition refers to the project definition phase, the time when a researcher decides the scope and objective of the project. There are eight parts to the definition phase:

(1) motivation	(5) perspective
(2) purpose	(6) domain
(3) object	(7) scope
(4) hypothesis	(8) importance

Motivation. There may be many motivations for an experiment on requirements tracing techniques. Researchers may be seeking to, for example,

Phase I: Definition	Phase II: Planning	Phase III: Realization	Phase IV:Interpretation
(1) Motivation Understand Improve Assess Validate Manage Assure Engineer Confirm Enhance Learn **(2) Purpose** Evaluate Test Implement Predict Characterize **(3) Object** Product Model Process Metric Theory **(4) Hypothesis** Null Hypothesis Alternative Hypothesis **(5) Perspective** Researcher Developer Maintainer Customer or User Project Manager **(6) Domain** Project Product Engineers **(7) Scope** Single Project Multi-Project Replicated Project Blocked Subject-project **(8) Importance: - Domain - Object of study** Safety-critical Mission-critical Quality of life Convenience	**(1) Experimental Design** Design Independent vars. - tracing technique - traceability data representation - traceability data mgmt - type of artifact - size of artifact - quality of artifact Dependent vars. - recall - precision - elapsed time Randomization Manipulation of independent variables **(2) Measurement** Metric definition: GQM FCM Metric validation Data collection: Automatability Form design and test Objective vs. subjective Scale: Nominal Ordinal Interval Ratio **(3) Product** Documentation Code Databases Other artifacts	**(1) Preparation** Pilot study Artifact development: Parsing reqts. Building answer sets Building thesauri Converting into input format **(2) Execution** Project execution Data collection Data validation **(3) Analysis** Quantitative vs. qualitative Preliminary data analysis Plots and histograms Model assumptions Primary data analysis Model application	**(1) Interpretation context** Statistical Framework Study purpose Field of research **(2) Results** Hypothesis Evaluation: Null hyp. rejected Null hyp. confirmed Acquisition of additional knowledge: Additional knowledge **(3) Extrapolation** Sample representativeness **(4) Impact** Visibility Replication Application

Table 1: Summary of our experimental framework.

- *understand* why certain elements are never traced to any other elements;
- *confirm* results that were seen on a previous experiment (of their own or by other researchers);
- *assess* a specific measure (e.g., recall) for a particular requirements tracing technique.

Purpose. The purpose of an experiment may be to:

- *test* a tool or specific implementation of an algorithm, e.g., test SuperTrace-Plus [21, 31];
- *evaluate* the effectiveness of a model or technique, e.g., evaluate the effectiveness of LSI when applied to the requirements tracing problem.

Other examples include, but are not limited to:

- *understand* a process or problem better;
- *improve* an existing tool or technique;
- *assess* the compliance of a tool or technique with a process, guideline, or criterion;
- *validate* the results of a previous experiment.

Object. The object of study will generally be a *product* or *model*, although some experiments will examine the requirements tracing *process* or the usefulness of a posited *metric*.

Hypothesis. The hypothesis (or hypotheses) should be stated in such a way as to be verifiable. The premise of the researcher, usually that "our new requirements tracing technique is better than someone else's technique as shown by higher recall and precision," will be stated as the *alternative hypothesis*. The *null hypothesis* will be that no difference between the requirements tracing techniques exists.

Perspective. Though most experiments are from the perspective of the *researcher*, they may be from many other perspectives such as *developer*, *maintainer*, *customer or user*, or *manager*.

Domain. The domains that typically comprise experiments are

- individual *engineers* who will be using the requirements tracing techniques, or
- *projects or programs (product)* on which the techniques will be applied.

Scope. Basili et al [7] classify experimental study scopes by looking at the size of the domains considered, as does this experimental framework. The following categories of experiments are considered:

- *Blocked subject-project* experiments examine one or more objects across a set of teams and a set of programs.
- *Replicated project* experiments look at objects across a set of teams and a single program.
- *Multi-project* variation experiments examine objects across a single team and a set of programs.
- *Single project* experiments look at objects on a single team and a single program.

Importance. We distinguish two levels of importance: domain importance and object of study importance. The former level assesses the importance of the domain of the experimental study, while the latter looks at the importance of the research being conducted (object of study). In both cases, the importance is being evaluated on the following scale:

- *safety-critical* (potential loss of human life),
- *mission-critical*,
- *quality of life*, or
- *convenience*.

For example, an experiment that evaluates a traceability model using requirements artifacts from an instrumentation and control system of a nuclear power plant will have *safety-critical* domain importance and *quality of life* object importance. In another example, an experiment evaluating IV&V analyst response to specific GUI features of a requirements tracing software tool using made-up data will have *quality of life* object importance and *convenience* domain importance.

4.2. *Planning*

The experiment planning phase consists of three parts:
(1) experimental design (2) measurement (3) product

Experimental Design. Experimental design has been addressed in detail in numerous works [7, 28, 8, 27] (just to mention a few). Here we concentrate on details of particular interest when performing requirements tracing experiments. A few definitions are required before proceeding. External validity refers to the generalizability of results. Internal validity refers to the believability of the relationship between the hypothesized causes and the experimental results. The independent variable is the factor that the researcher hypotheses will cause the results of the experiment. Experiments will be designed in such a way as to maximize internal and external validity, while evaluating the hypotheses. Designs range from incomplete block, complete block, to fractional factorial and full factorial. Treatment of these is beyond the scope of this paper. The interested reader should consult one of the many useful sources of information [7, 28, 8, 27, 26, 40].

In requirements tracing experimentation, the *requirements tracing technique* is the primary independent variable that determines the external validity of the class of experiments [28]. The representation used for the traceability data and the management of such data are also options. Other possible independent variables include the *type* and *size* of programs or project artifacts that are being traced, as well as the *quality* of these artifacts. Let us examine each of these in turn.

Requirements tracing technique. This will typically be an algorithm, tool, or process. Examples include vector space model, grep tool, commercially available requirements tracing tool, manual tracing process. Researchers will, on occasion, examine a more detailed application of a technique. For example, a researcher may examine the application of a threshold of 80% to the tracing results from a latent semantic indexing model.

Type and size of project artifacts. As presented in [24], *scalability* is the measure of the size of a dataset used for experimentation. Specifically, scalability is the extent to which the requirements tracing tool is able to achieve accuracy for "small" tracesets as well as "large" tracesets. A traceset typically consists of two artifacts that can be divided into lower level elements along with an answerset (a mapping between the two artifacts that has been validated). Hayes et al [24] define a "small" traceset to constitute 3000 combinatorial links or less. For example, a traceset consisting of 20 high level requirements and 50 low level requirements would have 20 x 50 = 1000 combinatorial links. Any traceset with more than 3000 combinatorial links is considered large. The average size of a requirement (typically measured as number of words) is of interest, but is rarely specified in research papers. The type of artifact is also of interest. Researchers have examined the tracing of code to user's manual pages (documentation), the tracing of one document level to another, etc. The type of element should also be specified – textual, source code, tabular, etc.

Quality of artifacts. Just as the number and type of defects in code or artifacts that are used to evaluate defect detection techniques are important, so is the quality of artifacts used for tracing experiments. If tracing experiments are only executed on artifacts that trace perfectly to each other (e.g., each high level requirement has at least one satisfying low level requirement, and vice versa), then the ability of a tracing technique to detect orphan low level requirements or unsatisfied high level ones cannot be validated. Besides ensuring that the dataset has at least some high level requirements with no matches in the low level and has some orphan low level requirements, the heterogeneity of the data must also be ensured. That is to say that there should be, if possible, some requirements that match a significant number of requirements as well as those that match just a few.

Dependent variables. In requirements tracing experiments, typical dependent variables are *recall*, *precision*, and *elapsed time for tracing*. As mentioned in Section 2.2, some other measures are also used on occasion as dependent variables. Randomization examines the assignment of subjects to the different levels of the independent variables [28]. Manipulation strategy refers to the combination of independent variables that have been studied [28]. For example, if the independent variables are technique (two are examined) and project (two are examined), a full factorial design would require that all levels of

both are crossed = technique x project = 2 x 2 or 4 trials.

Measurement. For this component of the planning stage, we have to specify the following components:

- definition of metrics (using, for example, goal-question-metric [9]),
- validation of metrics,
- collection of metrics (automatable or not),
- objectivity of metrics,
- scale of metrics (nominal/classificatory, ordinal, interval, or ratio) [7].

Product. The planning product section covers documentation, code, databases, and other artifacts. In some experimental studies, products are actually developed. For example, a software engineering experiment might have one team of developers build a system to a specification using an experimental development approach while another team uses a control approach. In traceability experiments, the products are usually the items that are being traced while a model or process is being evaluated.

4.3. *Realization*

The realization phase is the time when the experiment is conducted. There are three parts to the realization phase:

(1) preparation (2) execution (3) analysis (optional)

Preparation. Preparation often includes a *pilot study* [7]. For example, a small dataset (perhaps 10 x 10) may be used to get initial results for a new tracing technique. In tracing experiments, preparation may include *preparation of project artifacts* such as

- parsing requirements from documents,
- building answer sets,
- building or extracting thesauri,
- converting data in appropriate input format,
- etc...

Execution. Execution covers data collection and validation. Generally, tracing experiments collect similarity measures between parent and child elements. These are compared to the answerset. The number of correct links found, the number of incorrect links returned, the number of links missed, and the number of links returned for each element are used to calculate recall and precision.

Analysis. The analysis component includes preliminary data analysis, plots and histograms, model assumptions, primary data analysis, and model application. Tracing experiments typically depict recall and precision as lineplots, sometimes plotting recall and precision, and sometimes cutpoints.

4.4. *Interpretation*

Interpretation refers to the time when the researcher derives a result from the experimental study. There are four parts to the interpretation phase:

(1) interpretation context	(3) extrapolation
(2) results	(4) impact

Interpretation context. Interpretation context is the environment/circumstances that must be considered when interpreting the results of an experiment. The possible contexts are (i) *statistical*, (ii) *framework*, (iii) *study purpose*, or (iv) *field of research*.

For example, if interpretation context is statistical then the power of the statistical technique must be considered [28]. If a power table reports that the combination of technique, significance value, and number of observations yields a power of 90%, then the technique will not detect significant differences that are less than $1 - 0.9 = 10\%$. [28].

Results. We separate results of the studies into two categories: hypothesis evaluation and acquisition of additional knowledge.

We expect the primary result of any study to be either *confirmation* or *rejection* of the null hypothesis. While it is true that in most published studies the result is the rejection of the null hypothesis in favor of the alternative, we expect that in a large number of cases such results come with caveats. For example, if a paper studies the application of two or more IR methods to the requirements tracing problem, null and alternative hypotheses are stated for each individual method, and confirmed or rejected independently. A result of such study then may be rejection of some null hypotheses and confirmation of some others.

In addition to evaluating hypotheses, research studies might lead to acquisition of some new knowledge, either from insight gained due to specific charactersitics of object of study, peculiarities in measurements that required extra analysis, or simply a noted feature of any of the framework components. For example, Hayes et al. note [23] that the performance of IR methods varies depending on whether or not the same technical lingo had been used in both documents being traced. Another example of such extra knowledge gained is an observation made in [23] that human analysts working with the results of software may throw away some true links, but almost never find links missing from the software suggestions.

Extrapolation. Extrapolation deals with sample representativeness. In most cases, the issue of concern for tracing experiments is the representativeness of the projects and artifacts examined with the tracing technique. This was discussed in experimental design above.

Impact. Impact pertains to the level of effect that a study has on a field of research and/or industry. The level of impact will vary depending on the activities that occur after the experiment. Possible impacts include, but are not restricted to

- *replication of the experiment by others,*
- *replication of another study,*
- *application of the results in industry,*
- *visible publishing/presenting of the results.*

and can occur in any combination. Some impacts can be reported in the study itself, some others, such as being replicated in another study, may occur some time after the publication.

Some level of replication has been seen in the tracing experiments. For example, Marcus and Maletic [29] used the same project artifacts as Antoniol et al [3]. Results have been applied by a number of the projects that participated in tracing experiments. Publication of results is occurring in this area in top conferences and journals.

5. A Categorization of Requirements Tracing Experiments

The framework from Table 1 is used to categorize recent experiments that examined requirements tracing techniques. The keywords from the framework are italicized. We conclude the section with an in-depth look at an experiment that we conducted at the University of Kentucky.

5.1. *Antoniol and Canfora and Casazza and De Lucia and Merlo (2002)*

Antoniol et al performed an experiment that compared two requirements tracing techniques for two case studies [3].

Motivation. The motivation was to improve traceability link recovery between code and documents.

Purpose, Object, Hypothesis, and Perspective. Antoniol et al [3] conducted a study whose purpose was to evaluate two information retrieval models (the object is a model) from the perspective of a researcher. The null and alternative hypotheses were not formally specified, but by implication they were:

Null hypothesis: The results of using Vector Space Information Retrieval Model and Probabilistic Information Retrieval Model to trace two case studies, as measured by recall and precision, will not vary from the results of using grep on the same two case studies.

Alternative hypothesis: The Vector Space Information Retrieval Model (VSIR) and Probabilistic Information Retrieval Model (PIR) will achieve better results, as indicated by higher recall and/or higher precision, when applied to two case studies than will the grep tool.

The two models evaluated by Antoniol et al [3] are:

- Probabilistic Information Retrieval Model (PIR) - see Section 2.
- Vector Space Information Retrieval Model (VSIR) - see Section 2.

The tool used as a baseline comparison is `grep`. `Grep` is a unix utility that assists a user in performing textual searches interactively.

Domain, Scope, Importance, and Experimental Design. The scope was blocked subject-project where two projects (from the program domain) of convenience importance were traced. The next element, object of study importance, was quality of life. The independent variables were traceability model (the two models described above) and artifact projects. The two projects that were examined are described below.

Library of Efficient Data types and Algorithms (LEDA): LEDA is a freely available C++ library of foundation classes developed and distributed by Max-Planck-Institut für Informatik, Saarbrücken, Germany. The code and documentation of release 3.4, consisting of 95 KLOC, 208 classes, and 88 manual pages, was analyzed [3, 5].

Antoniol et al

Project:	LEDA	Albergate
Model:	PIR VSIR	PIR VSIR
Baseline:	`grep`	`grep`

Marcus and Maletic

Project:	LEDA	Albergate
Model:	LSI	LSI
Baseline:	PIR VSIR	PIR VSIR

Table 2: Experimental design for Antioniol et al[3] and Marcus and Maletic[29].

Antoniol et al

	LEDA [Pr, Rec]	Albergate [Pr, Rec]
PIR	[38.94%, 82.65%]	[34.16%, 70.68%] [13.8%, 100%]
VSIR	[17.06%, 72.44%]	[43.33%, 50%] [13.8%, 100%]

Marcus and Maletic

	LEDA [Pr, Rec]	Albergate [Pr, Rec]
LSI	[11.79%, 100%] [53.98%, 83.33%]	[16.38%, 100%] [21.12%, 85.96%]
PIR	[38.94%, 82.65%]	[34.16%, 70.68%] [13.8%, 100%]
VSIR	[17.06%, 72.44%]	[43.33%, 50%] [13.8%, 100%]

Hayes et al.

	MODIS [Pr, Rec]
VSIR	[11.4%, 25.4%]
VSIR+Thes.	[40.6%, 85.4%]
Analyst+STP	[46.15%, 43.9%]
STP	[38.8%, 63.41%]

Table 3: Comparison of results for Antioniol et al[3] and Marcus and Maletic[29] and Hayes et al.[23].

Albergate: Albergate is a software system, developed in Java, designed to implement all the operations required to administer and manage a small/medium size hotel (room reservation, bill calculation, etc.). It was developed from scratch by a team of final year students at the University of Verona (Italy) on the basis of 16 functional requirements written in Italian (as well as all other system documentation). Albergate consists of 95 classes and about 20 KLOC and exploits a relational database. Antoniol et al focused on the 60 classes implementing the user interface of the software system [3].

The dependent variables were recall and precision. Table 2 depicts Antoniol et al's experimental design. The first column of the table describes the information for the LEDA project or dataset, the second column pertains to the Albergate dataset. There was no randomization. Both tracing models were applied to both of the projects – full factorial design.

Measurement and Product. The recall and precision metrics are formally defined, validated metrics from the information retrieval field. The metrics were collected in an automated fashion and are ratio. The products were documentation and code.

Preparation, Execution, and Analysis. No pilot study was discussed. The artifacts were described by the authors as follows:

> "the LEDA manual pages contain a high number of identifiers that also appear in the source code. Actually, the LEDA team generated manual pages with scripts that extract comments from the source files. A markup language was used to identify the comment fragments to be extracted. Function names, parameter names, and data type names contained in these comments appear in the manual pages, thus making the traceability link recovery task easier. [3]"

For Albergate, source code classes were traced to functional requirements with the focus being on the 60 classes implementing the user interface of the software system [3]. The data collected was number of correct links found, the number of incorrect links returned, the number of links missed, and the number of links returned. Recall and precision were plotted. Table 3 shows exemplary results obtained in the experiments. The results are presented in a form of a pair of numbers: first number is *precision* and second number is *recall.* The study compared the results to using `grep`, however [3] provides the statistics on `grep` returning empty results rather than precision-recall numbers.

Interpretation. The interpretation context is the field of tracing research. The hypothesis result was that the null hypothesis was rejected in support of the alternative hypothesis. Other knowledge acquired included the discovery that "smoothing gives very low nonzero probabilities to unseen words; as a result, sometimes, a query is killed by the weight of word unseen in the training material [3]." The samples used are representative of the artifacts that are traced in practice in industry. This work did not replicate any prior experiments.

5.2. *Marcus and Maletic (2003)*

Marcus and Maletic [29] performed an experiment that applied one requirements tracing technique to the same two case studies used by Antoniol et al [3].

Motivation. The motivation was to improve traceability link recovery between code and documents.

Purpose, Object, Hypothesis, and Perspective. Marcus and Maletic [29] conducted a study whose purpose was to evaluate an information retrieval model (the object is a model) from the perspective of a researcher. The hypotheses used in the work, though not explicitly stated,

Null hypothesis: When applying LSI and PIR and VSIR, there is no difference in the precision and recall.

Alternative hypothesis: LSI will perform at least as well as PIR and VSIR in terms of precision and recall.

The model evaluated is:

- Latent Semantic Indexing (LSI) - see Section 2.

Domain, Scope, Importance, and Experimental Design. The scope was blocked subject-project where two projects (from the program domain) of convenience importance were traced. The next element, object of study importance, was quality of life. The experimental design is depicted in Table 2. The independent variables were model (LSI) and artifact projects. The first column of the table describes the information for the LEDA project or dataset, the second column pertains to the Albergate dataset. The two projects that were examined were described above in Section 5.1. The dependent variables were recall and precision. There was no randomization. Both projects were examined with the LSI technique.

Measurement and Product. The recall and precision metrics are formally defined, validated metrics from the information retrieval field. The metrics were collected in an automated fashion and are ratio. The products were documentation and code.

Preparation, Execution, and Analysis. No pilot study was discussed. The artifacts were described. The `man` pages of LEDA and Albergate were discussed in Section 5.1. The data collected was number of correct links found, the number of incorrect links returned, the number of links missed, and the number of links returned. Recall and precision were plotted. Table 3 shows exemplary results obtained in the experiments. The results are presented in a form of a pair of numbers: first number is *precision* and second number is *recall.* The study compared the results to those of [3].

Interpretation. The interpretation context is the field of tracing research. The alternative hypothesis was supported and the null hypothesis was rejected. Other knowledge acquired was that recall could be improved by using structural information of the C/C++ code. Often, classes were implemented in more than one file. Retrieving only one of them resulted in high precision but low recall [29]. The samples used are representative of the artifacts that are traced in practice in industry. This work used the same samples as Antoniol et al [3] but with a different technique.

5.3. *Antoniol, Caprile, Potrich, and Tonella (1999)*

Antoniol, Caprile et al [4] performed an experiment that examined a process for recovering "as is" design from code, comparing recovered design with the actual design and helping the user to deal with inconsistency [4].

Motivation: The motivation was to improve traceability recovery between code and "as is" design.

Purpose, Objective, Hypothesis, and Perspective: Antoniol, Capril et al [4] conducted a study whose purpose was to evaluate a process (object is process) from the perspective of a researcher. The null and alternative hypotheses were not formally specified, but by implication they were:

Null hypothesis: A tracing process consisting of distance computation and maximum match algorithm will not assist with design recovery as shown on an industrial telecommunications project.

Alternative hypothesis: A process consisting of distance computation and maximum match algorithm will assist with design recovery as shown on an industrial telecommunications system.

The process evaluated consisted of a number of steps: code and Object Model Technique (OMT) [36] design is translated to Abstract Object Language (AOL) using a tool; AOL is parsed to produce an Abstract Syntax Tree (AST) by a tool; a relations traceability check is performed; a dictionary traceability check that computes edit distance between attribute names is performed; a maximum matching algorithm and maximum likelihood classifier is applied; and results are displayed visually [4].

Domain, Scope, Importance, and Experimental Design: The scope was single project where one project of mission critical importance was traced. The next element, object of study importance, was quality of life. The independent variables were traceability process and artifact project. The project evaluated was an industrial telecommunications system and consisted of 29 C++ components, about 308 KLOC, for which object oriented object models and code was available [4]. The dependent variables were recall, precision, and average similarity. There was no randomization. The single project was examined with the tracing process.

Measurement and Product. The recall and precision metrics are formally defined, validated metrics from the information retrieval field. Average similarity is calculated by using the edit distance of attribute names found in the code and design. It is 0 when two strings have no characteristic in common and 1 when they coincide, hence it is a real value between 0 and 1 [4]. The metrics were collected in an automated fashion and are ratio. The products were design and code.

Preparation, Execution, and Analysis. No pilot study was discussed. The artifacts were described. Internal object models from a commercial computer-aided software engineering (CASE) tool are converted into AOL using a tool developed by Antoniol, Caprile, et al [4]. The internal models include class models, class relationships (such as aggregation and association). The other artifact traced was the C++ code corresponding to the internal object models. The data collected was average similarity, deleted classes (unmatched classes when performing the traceability check), true positives (number of correct links found), false positives (number of incorrect links returned), false negatives (number of links not returned that should have been), true negatives (number of links not returned that do not exist, i.e., true traceability errors in the artifact). Tables of average similarities and deleted classes as well as precision and recall were provided. Misclassification error was plotted. Code identifiers correctly segmented by design dictionary were plotted. Recall and precision were plotted.

Interpretation. The interpretation context is the field of tracing research. The hypothesis result was that the null hypothesis was rejected in favor of the alternative hypothesis. Other knowledge acquired included discovery that the "words used by the designer to build identifiers also make up the dictionary used in the code, but with some extensions. [4]" The samples used are representative of the artifacts that are traced in practice in industry. It appears that the process is still being applied by the telecommunications system project.

5.4. *Hayes, Dekhtyar, and Osborne (2003)*

Hayes, Dekhtyar, and Osborne performed an experiment that compared four requirements tracing techniques for one case study [23].

Motivation. The motivation was to improve traceability link recovery between hierarchical levels of textual requirements documents.

Purpose, Object, Hypotheses, and Perspective. Hayes, Dekhtyar, and Osborne [23] conducted a study whose purpose was to evaluate two information retrieval algorithms (the object is an algorithm) from the perspective of a researcher. The null and alternative hypotheses were not formally specified, but by implication they were:

Null hypothesis: The results of using VSIR and VSIR enhanced with a simple thesaurus algorithms to trace a case study, as measured by recall, precision, and performance, will not vary from the results of an analyst manually performing a trace of the same case study or of the analyst using the SuperTracePlus tool[21, 31] on the same case study.

Alternative hypothesis: The results of using VSIR and VSIR enhanced with a simple thesaurus algorithms to trace a case study, as measured by recall, precision, and performance, will be better than the results of an analyst manually performing a trace of the same case study or of the analyst using the SuperTracePlus tool[21, 31] on the same case study.

The methods, on which the algorithms evaluated by Hayes, Dekhtyar, and Osborne [23] are based, are

- Vector Space Information Retrieval (VSIR) – see Section 2.
- VSIR with simple thesaurus (VSIR+Thesaurus) – see Section 2.

The baseline comparisons are described below:

Analyst performing Manual Trace (AMT): The analyst used interactive searches in order to associate high level requirements and low level requirements.

SuperTracePlus (STP): This refers to results obtained from the requirements tracing module of SuperTracePlus (STP). STP, developed by Science Applications International Corporation (SAIC), uses keyphrase matching to generate candidate links. It is written in VBasic and Microsoft Access macros. The analyst may specify matching thresholds, e.g. 33%, 50%, etc. For example, if a high level requirement has four keyphrases and a low level requirement has two of these same keyphrases, a matching threshold of 50% would ensure that the low level requirement is returned in the candidate link list.

Analyst using SuperTracePlus (A&STP): The analyst examined the results returned by STP and made judgments on what constituted correct links or not and whether they needed to look for any more links.

Domain, Scope, Importance, and Experimental Design. The scope was single project where one project (from the program domain) of quality of life importance was traced. The next element, object of study importance, was quality of life. The independent variables were traceability algorithm (VSIR, VSIR+Thesaurus) and artifact project. The project that was examined was a NASA science instrument project, a moderate resolution imaging spectroradiometer (MODIS), with 19 high level requirements from [20] and 50 low level requirements from [19]. A typical requirement is one to two sentences in length. A sample requirement is:

> `''[The software] shall unpack all radiance data from 12-bits in the MODIS_pkt to Unpacked_MODIS_radiance when the packet contains radiance data, using the format documented in SBRS CDRL 305''`[19].

The dependent variables were recall, precision, and performance. There was no randomization. The single project was examined with the tracing algorithms.

Measurement and Product. The recall and precision metrics are formally defined, validated metrics from the information retrieval field. Performance was measured in hours. The former two metrics were collected in an automated fashion and are ratio. The latter metric was manually tracked and is ratio. The products were two levels of documentation.

Preparation, Execution, and Analysis. No pilot study was discussed. The artifacts were not described, but an example was given above. The data collected was number of correct links found, the number of incorrect links returned, the number of links missed, the number of links returned, and time to perform the trace (in hours). Recall, precision, and performance were compared in a tabular format. Table 3 shows the results obtained in the experiments. The results are presented in a form of a pair of numbers: first number is *precision* and second number is *recall*. The numbers are provided for VSIR and VSIR+Thesaurus as well as for the baseline cases: SuperTracePlus and Human Analyst+SuperTracePlus. In general, baseline methods can be seen to outperform VSIR, however VSIR+Thesaurus outperforms the baseline methods. In addition, VSIR and VSIR+Thesaurus algorithms were much faster, as is to be expected.

Interpretation. The interpretation context is the field of tracing research. The study confirmed the null hypothesis for VSIR algorithm and rejected it in favor of the alternative hypothesis for VSIR+Thesaurus. Among the other knowledge acquired during the study was the observation that the poor performance of the VSIR method was due to significant difference in technical lingo used in the high and low level requirements documents. The samples used are representative of the artifacts that are traced in practice in industry. This work did not replicate any prior experiments. After the study had been completed, a prototype software package called RETRO (REquirements TRacing On-target) was built [24], incorporating the algorithms tested. Also, the IR method toolbox of RETRO has been integrated with STP used by SAIC.

5.5. *Comparison of the Studies*

We have summarized the descriptions of the four studies [3, 29, 4, 23] in Table 4. From the broadest perspective possible, one can see from the table that this is an emerging field of research. Most studies are performed from the researcher's perspective, the objects of study are algorithms and models. The domain is almost always a program. Hypotheses are never explicitly stated, although they can always be determined. We will first examine the studies in detail, examining similarities and differences. We then identify areas that should be examined by future studies. Finally, we suggest some directions for our field in order to move beyond emerging research into more "mature" research.

It is clear from the table that all the studies examined shared common motivation and purpose. The objects of study differed for the studies, though there were basically only two categories addressed: tools (models or algorithms) and processes. The implicit hypotheses of the three studies [3, 29, 23] addressed the same idea, that specific IR methods (VSIR, PIR, LSI) may offer hope for improving requirements tracing. The studies all shared the same perspective of researcher. The domain differed for the studies, though project and program are very similar. The scope of the studies was evenly divided between single projects and blocked subject-project. The domain importance covered all but one of the possible choices yet object importance was quality of life for all studies.

As with domain, the independent variables varied, but not significantly. Three of the four studies examined traceability models or algorithms. The dependent variables were very similar for all studies with the exceptions being the addition of average similarity and performance for two of the studies. The product was the same for two of the studies. Preparation involved preparation of artifacts for all four studies, though the artifacts varied from open source artifacts to industry proprietary code and models. Interpretation context, results, and extrapolation were the same for all four studies. Impact ranged from studies that replicated other studies to studies whose results and tools are being utilized by industry now.

From the above, several observations can be made. First, by viewing the experiments in the framework, several patterns become evident. For example, the phrase " No pilot study was discussed" occurs repeatedly. Perhaps traceability researchers should consider performing small pilot studies prior to undertaking larger experiments. The phrase "The artifacts were described" also occurs frequently. This is a step in the right direction, but it would be more useful to other researchers if examples of the artifacts were shown in the paper and/or the artifacts were made available on-line. Second, it appears that other purposes might be considered when planning studies. For example, researchers might test specific tracing tools, improve existing algorithms, etc. Third, other objects might be studied. For example, a comprehensive study of *metrics* and their use/meaning/usefulness w.r.t. evaluation of tracing processes might be warranted, especially considering that new metrics[24] have been proposed recently. From the examination of the hypotheses, it appears that VSIR and PIR should be considered as baseline tracing methods for comparison purposes. It does not appear that methods such as grep need be examined further. Manual tracing methods cannot be dismissed though, as we require these for the human judgment task of the tracing process.

It appears that other perspectives should be considered in future traceability studies, such as project manager, developer, or customer. Studies should be undertaken that have safety-critical domain importance. Products of the studies are already diverse, but should explore other areas too such as non-textual artifacts. We should strive for all of our studies to be replicated and for the technologies under study to be adopted by industry.

As pointed out above, it appears that this field of research is emerging. To understand how we might move forward, let us consider the characteristics of a more mature field of software engineering, such as reading techniques. There, experimental studies are performed from numerous perspectives, such as project manager or maintainer. Studies have moved beyond baseline method comparisons to comparison of field-tested, proven techniques. In particular, the methods have been field tested and proven and have often been implemented in "productized" tools used in industry. Many studies have been replicated. Industry has adopted many of the studied techniques and tools. A community of researchers studying these techniques has been formed and is successfully collaborating with practitioners in the field.

Based on this brief analysis, the work that is before us in the requirements tracing and traceability research area is clear. We need to move beyond baseline methods such as VSIR. We need larger, standardized, more robust datasets (with answersets) available for study. We need to study the human factors associated with the tracing process (study from different perspectives, study different objects, study with different motivations and purposes). Finally, we need industry to be more actively involved with tracing/traceability research to facilitate large scale studies of the human factors in tracing.

6. Conclusions and Future Work

In this paper we presented a framework for characterizing experiments that examine requirements tracing techniques. The framework should assist researchers in developing and conducting additional experiments of this type. It also facilitates the comparison of results from similar experiments. We used the framework to describe and compare four recent experimental studies. We used the framework to identify areas for future research as well as for future experimentation. We also identified suggestions for moving tracing research from an emerging field to a more mature field.

We have been actively pursuing these suggestions in our own work. We have developed a prototype tool that is being used by industry. We have experimented on a number of new, larger programs. We have developed additional measures. We plan to enhance the prototype tool that we have developed in order to productize it, and we plan to conduct human factors studies.

To encourage the replication of the experiment performed at the University of Kentucky, the dataset used along with the answer set has been posted on the Software Engineering Experimentation Web (SEEWeb) hosted by George Mason University at http://ise.gmu.edu:8080/ofut/jsp/seeweb/index.jsp. Though the experiments presented here all achieved fairly consistent results in terms of recall and precision, replication of experiments can only serve to strengthen the results.

7. Acknowledgements

Our work is funded by NASA under grant NAG5-11732. Our thanks to Ken McGill, Tim Menzies, Stephanie Ferguson, Mike Chapman and the Metrics Data Program, and the MODIS project for maintaining their website that provides such useful data. We also thank our current and former students James Osborne, Senthil Sundaram, Ganapathy Chidambaram and Sarah Howard for their participation in the requirements tracing research. Without them, this work would not be possible. We also thank Massimiliano di Penta and Jonathan Maletic for enlightening discussions about their research and ours.

1. M. Abrahams and J. Barkley. Rtl verification strategies. In *PIEEE WESCON/98*, 1998.

Phase I: Definition				
	ANT02[3]	MM03[29]	ANT99[4]	HAY03[23]
Motivation	*improve traceability*			
Purpose	*evaluate*			
Object	*models*		*process*	*algorithms*
Null Hypothesis	VSIR, PIR – same as grep	LSI – same as VSIR, PIR	Edit distance, Max. matching alg., max. likelihood – don't help tracing	VSIR, – VSIR+Thesaurus – same as human, STP
Alt. Hypothesis	VSIR, PIR – better than grep	LSI – better than VSIR, PIR	Edit distance, Max. matching alg., max. likelihood – help tracing	VSIR, – VSIR+Thesaurus – better than human, STP
Perspective	*researcher*			
Domain	*program*		*project*	*program*
Scope	*blocked subject-project*		*single project*	
Dom. Importance	*convenience*		*mission-critical*	*quality of life*
Obj. Importance	*quality of life*			
Phase II: Planning				
	ANT02	MM03	ANT99	HAY03
Ind. Variables	traceability model; artifact projects		traceability process	traceability algs.
Dep. Variables	recall; precision		avg. similarity	performance
Product	Documentation and code		Design and code	Two levels of documentation
Phase III: Realization				
	ANT02	MM03	ANT99	HAY03
Preparation	LEDA, Albergate `man` pages, code		CASE object models, code	MODIS textual requirments
Phase IV: Interpretation				
	ANT02	MM03	ANT99	HAY03
Context	tracing/traceability research field			
Results	null hypothesis rejected in favor of alternative			null hyp. confirmed for VSIR, rejected for VSIR+Thesaurus
Extrapolation	industry representative			
Impact	not replication, was replicated	partial replication of ANT02	not replication industry now using process	

Table 4: Comparison of the four studies described in this paper.

2. D. Anezin. Process and methods for requirements tracing (software development life cycle), 1994.
3. G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, 2002.
4. G. Antoniol, B. Caprile, A. Potrich, and P. Tonella. Design-code traceability for object oriented systems. *Annals of Software Engineering*, 9:35–58, 1999.
5. Algorithmic Solutions Software GmbH (AS). Leda research. http://www.mpi-sb.mpg.de/LEDA/leda.html.
6. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, Addison-Wesley, 1999.
7. V. Basili, R. Selby, and D. Hutchens. Experimentation in software engineering. *IEEE Transactions on Software Engineering*, 12(47):733–743, 1986.
8. V. Basili, F. Shull, and F. Lanubile. Building knowledge through families of software studies: An experience report. *IEEE Transactions on Software Engineering*, 25(4):456 – 473, 1999.
9. V.R. Basili and H.D. Rombach. The tame project: Towards improvement-oriented software environments. *IEEE Transactions on Software Engineering*, 14(6):758–773, 1988.
10. S. Bohner. A graph traceability approach for software change impact analysis, 1995.
11. P. Bourque and A. Abran. An experimental framework for software engineering research. In *Proceedings of the Forum On Software Engineering Standards Issues 1996*, 1996.
12. P. Brouse. A process for use of multimedia information in requirements identification and traceability, 1992.
13. A. Casotto. Run-time requirement tracing. In *Proceedings of the IEEE/ACM International Conference on Computer-aided Design*, 1993.
14. J. Cleland-Huang, C.K. Chang, G. Sethi, K. Javvaji, H. Hu, and J. Xia. Automating speculative queries through event-based requirements traceability. In *Proceedings of the IEEE Joint International Requirements Engineering Conference (RE '02)*, pages 289–296, 2002.
15. F. Crestani, M. Lalmas, C. J. van Rijsbergen, and Iain Campbell. Is this document relevant? ... probably": A survey of probabilistic models in information retrieval. *ACM Computing Surveys*, 30(4):528–552, 1998.
16. S. Deerwester, S.T. Dumais, T.K. Landauer, G.W. Furnas, and R.A. Harshman. Indexing by latent semantic analysis. *Journal of the Society for Information Science*, 41(6):391–407, 1990.
17. N. Fenton, S. Pfleeger, and R. Glass. Requirements tracing. *IEEE Software*, July 1994, 1994.
18. Orlena Gotel and Anthony Finkelstein. Extended requirements traceability: Results of an industrial case study. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, page 169. IEEE Computer Society, 1997.
19. GSFC SBRS. Level 1a and geolocation processing software requirements specification.
20. GSFC SBRS. MODIS science data processing software requirements specification vesion 2.
21. J. Huffman Hayes. Risk reduction through requirements tracing. In *Conference Proceedings of Software Quality Week 1990*, 1990.
22. J. Huffman Hayes. Energizing software engineering education through real-world projects as experimental studies. In *Proceedings of the Conference on Software Engineering Education and Training, CSEET*, 2002.
23. J. Huffman Hayes, A. Dekhtyar, and J. Osbourne. Improving requirements tracing via

information retrieval. In *International Conference on Requirements Engineering, Monterey, California*, pages 151–161, 2003.

24. J. Huffman Hayes, A. Dekhtyar, S. Sundaram, and S. Howard. Helping analysts trace requirements: An objective look. In *International Conference on Requirements Engineering (RE'2004)*, 2004.
25. Jane Hayes, Alex Dekhtyar, and Senthil Sundaram. Measuring the effectiveness of retrieval techniques in software engineering. Technical Report TR 422-04, University of Kentucky, October 2004.
26. N. Juristo and A. Moreno. *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, 2001.
27. F. Lanubile. Empirical evaluation of software maintenance technologies. *Empirical Software Engineering: An International Journal*, 2(2):95–106, 1997.
28. C. M. Lott and H. D. Rombach. Repeatable software engineering experiments for comparing defect-detection techniques. *Empirical Software Engineering: An International Journal*, 1(3):241–277, 1996.
29. A. Marcus and J. Maletic. Recovering documentation-to-source code traceability links using latent semantic indexing. In *Proceedings of the Twenty-Fifth International Conference on Software Engineering 2003*, pages 125–135, 2003.
30. J. Matthias. Requirements tracing. *Communications of the ACM*, 41(12), 1998.
31. T. Mundie and F. Hallsworth. Requirements analysis using supertrace pc. In *Proceedings of the American Society of Mechanical Engineers (ASME) for the Computers in Engineering Symposium at the Energy & Environmental Expo 1995*, 1995.
32. R. Pierce. A requirements tracing tool. In *Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues*, 1978.
33. B. Ramesh. Factors influencing requirements traceability practice. *Communications of the ACM*, 41(12):37–44, 1998.
34. B. Ramesh and M. Jarke. Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*, 27(1):58–93, 2001.
35. S. E. Robertson. The probabilistic character of relevance. *Information Processing & Management*, 13(4):247–251, 1977.
36. J. Rumbaugh, M. Blah, Premerlani W, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.
37. G. Spanoudakis, A. Zisman, E. Perez-Minana, and P. Krause. Rule-based generation of requirements traceability relations. *Journal of Systems and Software*, 72(2):105–127, 2004.
38. T. Tsumaki and Y. Morisawa. A framework of requirements tracing using uml. In *Proceedings of the Seventh Asia-Pacific Software Engineering Conference 2000*, pages 206–213, 2000.
39. R. Watkins and M. Neal. Why and how of requirements tracing. *IEEE Software*, 11(4):104–106, 1994.
40. C. Wohlin, P. Runeson, M. Host, M.C. Ohlsson, B. Regnell, and A. Wesslon. *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers, 2000.
41. A. Zisman, G. Spanoudakis, E. Perez-Minana, and P. Krause. Towards a traceability approach for product family requirements. In *Proceedings of the 3rd ICSE Workshop on Software Product Lines: Economics, Architectures and Implications*, 2002.