ISR Institute for Software Research
University of California, Irvine

Establishing the Connection Between Software Traceability and Data Provenance

Hazeline Asuncion
University of California, Irvine
hasuncio@ics.uci.edu

Richard N. Taylor
University of California, Irvine
taylor@ics.uci.edu

November 2007

ISR Technical Report # UCI-ISR-07-9

Institute for Software Research
ICS2 217
University of California, Irvine
Irvine, CA 92697-3455
www.isr.uci.edu

Establishing the Connection Between Software Traceability and Data Provenance

Hazeline Asuncion and Richard N. Taylor
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3425
{hasuncion, taylor}@ics.uci.edu

ISR Technical Report # UCI-ISR-07-9

November 2007

Abstract:

Researchers and practitioners alike agree that software traceability is important to software development. Despite its recognized utility, software traceability has largely been infeasible in practice due to the high costs involved and the low benefits obtained. In the first part of this survey, we identify the difficulties that hinder end-to-end software traceability, and we analyze these difficulties from economic, technical, and social perspectives. We also discuss current approaches that attempt to address the identified difficulties. In the second part of this survey, we highlight striking similarities between software traceability and the concept of data provenance in e-Science. We investigate whether data provenance techniques can potentially address the difficulties of implementing end-to-end software traceability. Inspired by data provenance techniques, we provide insights for improving software traceability.

Table of Contents

Establishing the Connection Between Software Traceability and Data Provenance

Hazeline Asuncion and Richard N. Taylor
Institute for Software Research

ISR Technical Report # UCI-ISR-07-9
November 2007

Abstract

Researchers and practitioners alike agree that software traceability is important to software development. Despite its recognized utility, software traceability has largely been infeasible in practice due to the high costs involved and the low benefits obtained. In the first part of this survey, we identify the difficulties that hinder end-to-end software traceability, and we analyze these difficulties from economic, technical, and social perspectives. We also discuss current approaches that attempt to address the identified difficulties. In the second part of this survey, we highlight striking similarities between software traceability and the concept of data provenance in e-Science. We investigate whether data provenance techniques can potentially address the difficulties of implementing end-to-end software traceability. Inspired by data provenance techniques, we provide insights for improving software traceability.

1 Introduction

Software traceability is important to the success of a software development project. Software traceability relates the various information products generated in software development, called artifacts, to enable a comprehensive understanding of the software being produced. The benefits of achieving software traceability include better verification and validation of customer requirements, lower maintenance costs, and better assessment of product quality.

Despite the benefits of software traceability, tracing artifacts across the entire software development lifecycle, or end-to-end traceability, is difficult to achieve. The confluence of factors such as the distribution of artifacts across different groups, the heterogeneity of artifacts and tools used, and the rapid changing nature of artifacts poses challenges to tracing artifacts. Because artifacts are distributed across different groups, artifacts are inaccessible and are difficult to trace. Because of the heterogeneity of artifacts, it is difficult to trace across multiple formats and across multiple levels of abstraction. Because of the heterogeneity of tools that lack interoperability, it is difficult to represent traceability links. Because of rapidly changing nature of artifacts, established trace links quickly become obsolete. All these factors contribute to the high cost of supporting traceability.

To find fresh approaches to achieving traceability, we seek insights from e-Science, a domain with characteristics similar to software engineering. Data provenance techniques in e-Science enable the tracing of data products across an entire experiment lifecycle. We analyze the ways

that data provenance systems can potentially fulfill the requirements necessary for successful end-to-end software traceability.

The survey is organized as follows. Section 2 discusses software traceability and analyzes the factors that hinder software traceability in practice. Section 3 identifies requirements for end-to-end software traceability. Section 4 introduces the domain of e-Science. Section 5 discusses the similarities between data provenance and software traceability and the rationale for examining data provenance techniques. Section 6 presents related works in software traceability and in data provenance. Section 7 surveys the ways that data provenance systems and techniques potentially meet the software traceability requirements. Insights gleaned from data provenance are presented in Section 8. Finally, Section 9 concludes with insights that are applicable to software traceability.

2 Software Traceability

This section provides a brief history and accepted definitions of software traceability in the literature. Next, we analyze the reported difficulties in achieving software traceability and we categorize them into three perspectives. Then, we briefly survey existing approaches that aim to address these problems. We end the section with our definition of software traceability.

2.1 Brief History

Historically, software traceability has predominantly been applied to the area of requirements engineering. Requirements traceability was introduced in the 1970s to minimize the drift between the software product's actual behavior and the original requirements specified by the customer [22]. Originally concerned with tracing between requirements artifacts, the field of software traceability has grown to accommodate other types of artifact relationships across the software lifecycle [103] with the goal of enhancing the software product quality. Almost 40 years after the concept of traceability was introduced in the field of software engineering, the research literature is replete with approaches to software traceability [27, 28, 51, 52, 59, 61-63, 86, 90, 95, 102]. However, the inability to achieve traceability still exists in industry [51, 74, 103].

2.2 Definitions

Definition: Software artifact is defined as "a piece of information produced or modified as a part of the software engineering process" [51]. Examples of software artifacts include requirements documents, design documents, code, and test cases.

Definition: Requirements traceability is defined as "the ability to describe and follow the life of a requirement, in both forwards and backwards direction" [68]. Requirements are the encoded customer expectations of a software product [96]. Requirements traceability is capturing the relationship between requirements artifacts and the other artifacts in the software lifecycle.

Requirements traceability has been classified into pre- and post- requirements specification traceability. Pre-requirements specification (pre-RS) traceability is concerned with tracing requirements to the sources of requirements while post-requirements specification (post-RS) traceability is concerned with tracing requirements to downstream artifacts in the lifecycle [68].

Definition: Software traceability is defined as the **"**ability to relate artefacts created during the development of a software system to describe the system from different perspectives and levels of abstraction with each other, the stakeholders that have contributed to the creation of the artefacts, and the rationale that explains the form of the artefacts" [103]. This broader definition of traceability encompasses the various possible relationships between artifacts.

Definition: Trace link represents a relationship between artifacts. A trace link may or may not contain the type of relationship represented, i.e. link semantics.

2.3 Benefits of Software Traceability

The importance of software traceability has been recognized by many researchers [59, 64, 74, 78, 111] since it aids the following activities: system comprehension, impact analysis, system debugging, and communication between the development team and stakeholders [59, 77, 90, 92, 95]. System comprehension is enhanced because software traceability connects the rationale (e.g. reasons for designing the system and reasons for using a component) to artifacts. Impact analysis is supported because software traceability identifies which parts of the software system are affected by a changed artifact. System debugging is informed because software traceability couples use cases with their implementation. Communication between the development team and stakeholders is facilitated because software traceability associates artifacts with the contributors of those artifacts. In some instances, traceability is needed to comply with internal standards and external regulations [32, 80, 92, 110]. Other benefits to traceability include lower maintenance costs and better assessments of product quality, both of which lead to improved customer relationships [73, 92].

2.4 Problem Analysis

The inability to achieve software traceability in practice, henceforth referred to as the traceability problem, still exists. Despite the numerous approaches suggested in the research literature, they are not adopted in practice [68, 103, 106]. Even in places where a requirements traceability approach is in place, the difficulties of tracing artifacts are still reported [32, 68, 103]. Since it is rare to achieve end-to-end traceability in practice [31], it has been identified as one of the grand challenges in traceability [74].

This section surveys the reported manifestations of the traceability problem in the literature. Since the traceability problem is multi-faceted [69], we examine it from three different perspectives: economic, technical, and social perspectives. Examining the software traceability problem from these three perspectives helps us to understand why software traceability is difficult to achieve in practice.

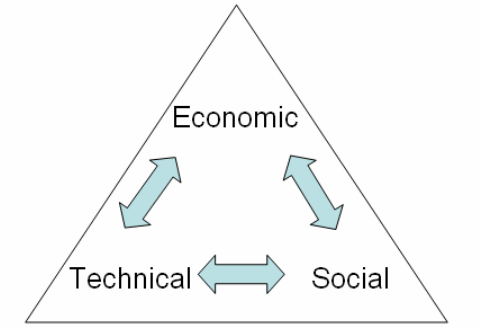

Figure 1: Traceability Perspectives [32]

2.4.1 Economic Perspective

The economic perspective focuses on the cost of supporting traceability. Implementing traceability has been known to incur high costs in terms of labor hours [77, 92] and high costs are a hindrance in achieving effective traceability [21]. There is a high cost associated with defining traceability links as well as maintaining these links [20]. For instance, a case study of a large government funded project reports that the costs of implementing traceability is more than double the normal documentation costs [92]. Some practitioners argue that time spent in performing traceability tasks could have been allocated to writing software code [54]. Even with companies that are willing to pay the high costs of traceability, the expected benefits are still not realized [93, 106]. Other sources of costs include purchasing or developing a trace tool as well as training users [32].

To mitigate the cost, some approaches examine the tradeoffs between cost and quality [63] or between cost and benefit [62]. Regarding the tradeoff between cost and quality, one can search for the optimal position by reducing the level of granularity of traces in order to save costs while still maintaining an acceptable level of quality [63]. For instance, generating trace links at the method source code level is more expensive than tracing at the class source code level even though there is usually not much difference in the accuracy of links; thus, in this case, one should not perform fine-grained tracing of the source code. Meanwhile, approaches that examine the cost-benefit tradeoff assign importance values to trace links and concentrate on tracing over only higher value links to minimize cost [62]. One limitation of this scheme is that values assigned to trace links are entirely project-specific. A similar approach allocates more trace support to the crucial parts of the software system [52].

2.4.2 Technical Perspective

The technical perspective deals with establishing and maintaining trace links as well as tracing across heterogeneous artifacts and heterogeneous tools. While the economic perspective addresses the costs in supporting traceability, the technical perspective addresses the complexity of tracing due to the explosion of the artifact space, the differing levels of formality of artifacts, and the numerous relationships [102], sometimes implicit [27], that occur at various levels of granularity. The different types of trace relationships are surveyed in [103].

Explosion of Artifact Space

Tracing across various artifacts in a software development lifecycle is difficult due to the sheer number of artifacts and the numerous relationships between these artifacts. While it is necessary to capture all relevant traces to avoid the loss of knowledge, capturing too many traces is unwanted. Excessive traceability is known to be unmanageable [59] and can negatively impact the accuracy of links [62]. Thus, it is important to know the boundary between complete and excessive tracing.

There are currently different ways of bounding the problem space of artifacts and artifact-relationships. The agile community advocates a lean traceability approach where the only traces captured are those determined to be relevant by the developers of the system [54]. This approach assumes that the developers already have a basic understanding of the system. The approach may also be subject to staff turnover. The selection of specific artifacts to trace can also be based on the project manager's discretion or the information gleaned from past projects [59]. However, this approach does not aid inexperienced managers or organizations that lack records of past projects. Some examples of the types of artifacts that may be captured are in [59].

Maintenance of Trace Links

Maintaining trace links is another major problem due to the prevalence of link deterioration. Link deterioration is defined as the obsolescence of links due to the evolution of artifacts. Artifacts evolve independently and the changes are not reflected in the trace links nor are they reflected in the related artifacts. For example, in a requirements-centric traceability, changing the requirements necessitates the update of all the corresponding links and related artifacts. Without a systematic approach to performing updates, the cost of maintaining traceability can be very high. Not only is the cost high, but there is also no guarantee that all the impacted links are updated. Thus, the volatility of requirements is identified as one of the main sources of difficulty in large complex systems [93].

Approaches to maintaining trace links include controlling artifact changes, cascading changes through events, and recovering traces automatically. Artifact changes can be controlled by establishing a development process to disallow people from changing artifacts directly (e.g. Review Boards [102]). In this approach changes have to be approved by a review board. Since this process imposes high overhead, only high visibility documents go through review boards. The cascading of artifact changes through events can be performed by event-based traceability, which uses the publish-subscribe mechanism to relate various artifacts to the requirements artifact [51]. Thus, when a requirements artifact changes, the subscribed artifacts are notified. A drawback of this approach is the cost of manually registering artifacts with the requirements artifact. The third approach is recovering candidate trace links automatically through information retrieval (IR) techniques. To date, trace recovery techniques have not been able to provide full accuracy [76]. One hindrance is that artifacts must be preprocessed before IR techniques are used [53]. Even with more sophisticated IR techniques, it is difficult to achieve high recall and precision rates. (Recall is the percentage of retrieved links out of all relevant links while precision is the percentage of correct links out of the retrieved links [54].)

Heterogeneity of Artifacts
The heterogeneity of artifacts is another factor that contributes to the traceability problem. Artifacts produced in the course of software development vary in their levels of formality, ranging from unstructured documents to highly formal code. The differing formats and notations by which artifacts are expressed as well as the different levels of abstractions represented by the artifacts present challenges to establishing traces across different artifact types.

Approaches that address the heterogeneity of artifacts include two-dimensional traceability, model transformation, and information integration. Two-dimensional traceability enables tracing between different types of artifacts to increase program comprehension [81]. The two dimensions, horizontal and vertical traceability, are complementary to each other. Horizontal traceability maps associated items across different artifacts while vertical traceability maps items within the same artifact. Another approach, model transformation, enables tracing design artifacts across multiple levels of abstraction. Transformations can vary in the level of automation. Fully automated transformation entails the use of transformation specification on a design artifact to produce a realization that is at a lower level of abstraction [23]. Still another approach, information integration, translates heterogeneous artifacts into a common format in a repository. Trace relationships between artifacts are automatically generated within the repository [27].

Heterogeneity of Tools
Tracing software artifacts across different tools is difficult due to the lack of interoperability between different tools [59, 68, 92]. The separation of information by tools is referred to as the "islands of information" problem [27]. For instance, changing the artifacts outside a trace tool does not guarantee that the artifacts inside the trace tool are updated [55, 59]. The lack of interoperability between different tools necessitates redundant data entry [31]. Not only is redundant data entry a tedious task, but it also adds the overhead of reconciling data [27].

One way to address tool heterogeneity is through the use of custom code to enable different tools to exchange data with a shared repository [32]. This approach avoids the problem of redundant data entry since artifact changes are always reflected in the shared repository.

2.4.3 Social Perspective

The social perspective is equally important to consider, since it focuses on the interaction of various groups and their effect on traceability. This perspective also focuses on the expectations and attitudes of various stakeholders toward traceability. It is recognized that the human element plays a crucial part in determining the quality of traceability [31, 74, 76]. For instance, regardless of the results of the automatically generation of trace links, users determine whether the trace links are correct in the end [76].

Different Groups Own Different Artifacts
Traceability across artifacts owned by different groups is difficult due to the lack of accessibility of artifacts to those outside the groups. For example, the lack of accessibility to the requirements' sources, which could be distributed among multiple groups, has been the most frequently cited problem by practitioners [68]. In addition, distributing the ownership of requirements among different groups makes it difficult to trace the dependency relationships

among the requirements [93]. Lack of communication between groups is one of the factors that contribute to the lack of accessibility of artifacts [68].

Approaches that address the lack of accessibility to artifacts include negotiating changes to upstream artifacts and publishing artifacts to a portal. Encouraging teams to negotiate the changes to upstream artifacts like requirements enhances communication between groups and increases the accessibility of artifacts [31]. Publishing artifacts to a portal raises the visibility and accessibility of artifacts to other groups [32].

Differing Expectations of Traceability Tool

Implementing software traceability is difficult since traceability some times carries different meanings to different people. For example, requirements traceability can mean tracking requirements in the contract for estimating project costs, tracking requirements to various artifacts in the lifecycle, or tracking the input and output of phases in the lifecycle [68]. In addition, stakeholders may have different expectations of trace tool [92]. For instance, a maintenance engineer expects support for impact analysis while a project manager expects support for tracking project status.

One way to address different stakeholder expectations is by identifying the key users of a trace tool and developing custom in-house extensions to existing trace tools [93] or developing their own custom trace tools [32].

Low Motivation for Performing Traceability Tasks

In general, software engineers have little or no motivation to perform traceability tasks [30, 77]. To them, traceability tasks are "laborious" [23] and "burdensome" [21]. In one study, half of the subjects who were commissioned to verify trace links dropped out because they "disliked" tracing [76]. There are several reasons for the low motivation of software engineers. One reason is that traceability tasks are additional imposed work with no direct benefits [31, 77], known as the Traceability Benefit Problem [31]. Other reasons include the lack of understanding of the usage of trace information and the lack of first-hand knowledge of the artifacts [31].

One way to address the low motivation for performing traceability tasks is by coupling traceability tasks with the usage [31]. Another method is to use trace information to directly support stakeholders in their lifecycle tasks [32] in order to provide direct benefits to users.

Other Difficulties

In addition to the above manifestations of the traceability problem, the following social factors are also reported in the literature: privacy [59, 92], politics [68, 77], low priority given to traceability [68] and unrecorded links due to lack of time [81]. One way to address privacy is by considering all artifacts as owned by a project team so that traces to individual contributors will not be used in performance evaluation [93].

2.4.4 Perspective Interplay

Factors that pose challenges to traceability are not in isolation. A factor affects or is affected by factors in other perspectives (See Figure 1). The following subsections illustrate the interplay between the different perspectives and explain why solely addressing one perspective falls short

of addressing the traceability problem. We do not provide an exhaustive list of all possible interactions, but simply illustrate the interplay between the different perspectives.

2.4.4.1 Interplay of Economic and Technical Perspectives

There is a bidirectional relationship between the economic and technical perspectives. The economic perspective is a major factor in determining whether a traceability approach will be adopted in industry [32]. The cost of establishing and maintaining trace links affects the number of artifact and relationship types that will be traced by an organization. Since fine-grained tracing is more costly [31, 34], the economic perspective also determines the level of granularity that will be traced.

Meanwhile, the technical perspective also affects the economic perspective. The level of tool support in establishing or defining traceability links heavily determines the cost of tracing [81]. The lack of interoperability between tools also contributes to the high cost of traceability since this necessitates redundant data entry and manual reconciliation [27, 32].

There is also tension between capturing all possibly relevant links to ensure that no loss of knowledge occurs [59, 93] and taking a minimalistic approach in trace capture [54] to lower the cost. There is currently a lack of cost-benefit models [74] that guide organizations in selecting the types of artifacts, the level of granularity, and the types of relationships to trace.

2.4.4.2 Interplay of Social and Technical Perspectives

There is also a bidirectional interaction between the social and technical perspectives. For instance, due to the low motivation of software engineers in performing traceability tasks, the captured traces were unusable in one case study [31]. In addition, different user expectations [68] make it difficult to use a commercial trace tool without customization. Meanwhile, the technical perspective also affects the social perspective. If a trace tool supports the development activities of stakeholders, it is more likely to be adopted [32, 86].

2.4.4.3 Interplay of Economic and Social Perspectives

There is also a relationship between the economic and the social perspectives. Due to the high costs required in performing traceability tasks, most software engineers have an aversion toward traceability [76, 77]. The high startup and maintenance cost of the manual approaches is also one of the common complaints of developers [30].

The social perspective also affects economic perspective. Lack of accessibility of artifacts between groups can make tracing across groups more costly since more time is spent locating artifacts.

2.4.4.4 Interplay of Economic, Technical and Social Perspectives

There is also interplay between the three perspectives. One example where this interplay can be illustrated is the automation of trace link generation. To mitigate the costs, information retrieval methods are used [29, 55, 75, 83] to provide automated support for traceability at the risk of

potentially establishing inaccurate links. Not only is this technique limited by its inability to provide fully accurate links, but it also does not provide the level of semantics needed for analysis [103]. Another difficulty with information retrieval techniques is that the accuracy is dependent on people [54, 55, 76]. First, the artifacts must be pre-processed to be more amenable to the information retrieval technique in order to yield better results [55]. Since link recovery techniques cannot reach 100% accuracy, post-processing by a human analyst identifies which of the candidate links generated are correct links [76]. Without human intervention, the validity of the traces is in question [62]. However, human analysts are also prone to error [76]. Hence, we see that all three perspectives can be interwined.

2.5 Working Definition of Software Traceability

As illustrated in the previous subsection, an effective end-to-end traceability approach should address the economic, technical, and social perspectives simultaneously. Our definition for software traceability tackles these three perspectives.

Definition: Software traceability elucidates relationships between artifacts in such a way that it supports various stakeholders in their software lifecycle tasks. An elaboration of this definition follows.

- **Elucidate relationships:** Software traceability should explain or make clear the relationships between artifacts through the captured trace semantics.
- **Support the entire software lifecycle:** Traceability should not be restricted to one phase in development, nor should it be restricted to development activities prior to deployment. Traceability should enable *comprehensive system understanding* by enabling traceability from software inception to retirement. For example, traceability should lower software maintenance costs by increasing the accessibility of related artifacts.
- **Support stakeholders in their software lifecycle tasks:** Various stakeholders have different notions of traceability and we aim to cater to these differing interests. The recorded traces should have semantic information beneficial to stakeholders. We use the *concept of layered traceability* where users can customize their traceability view by maintaining their customized trace information. This also implies the ability to trace at different levels of abstraction. Since the usage of traceability is stakeholder-defined, traceability is not merely an overhead task to satisfy external (e.g. customer) requirements. Instead, traceability supports stakeholders in their day-to-day tasks. By enabling stakeholders to directly use the traced information they capture, the accuracy of traces increases. Compliance with external traceability requirements is simply a by-product of maintaining accurate internal (i.e. within project team) trace information. Supporting life cycle tasks implies that a benefit is derived from the cost of establishing and maintaining traceability. The cost is invested into supporting actual development tasks.

We have just examined the software traceability problem from the economic, technical, and social perspectives. We also discussed our definition of software traceability. We will now move on to discuss our framework of comparing provenance systems. This framework addresses the specific challenges to traceability identified in Section 2.4.

3 Requirements for Effective End-To-End Software Traceability

This section discusses important traceability properties that are necessary to achieve successful end-to-end traceability. The following properties address the specific difficulties in achieving traceability (see Section 2.4 and [18, 74]). The table below maps the properties to the primary perspective that they address.

	Economic	Technical	Social
Capturability		X	
Utilizability	X		
Affordability	X		
Maintainability		X	
Accessibility			X
Scalability		X	
Customizability			X
Auditability		X	

Table 1: Perspectives that the Software Traceability Properties Address

3.1 Capturability

One of the identified problems with software traceability lies in establishing traceability links. Manual approaches enable semantically-rich links to be captured but at a high cost. On the other hand, trace recovery techniques lower cost but the links do not deliver the right level of semantics [103]. Thus, capturability is the property concerned with the means of capturing traces as well as the type of information captured. This property distinguishes between automatic and manual trace capture since the type of capture affects the level of manual intervention required. Finally, since the level of granularity can affect both the cost of tracing and the understandability of traces, it is important to understand the level of granularity supported.

Capture Trace Semantics

A sub-property of capturability, relationship semantics is concerned with capturing the semantics of trace links. If the semantics are not explicitly recorded, they can be inferred via **context**. In this survey, context is defined as the following:

Definition: Context refers to the surrounding processes, people, tools or artifacts that affect either the production or modification of a given artifact. Context aids in understanding trace relationships, since it can provide information on how and why artifacts are related. However, the difficulty in a typical software development project is that context is lost.

3.2 Utilizability

Utilizability is a property concerned with the usage of trace information, e.g. querying and applying trace information to accomplish other tasks. This property also identifies the actual benefits gained in having the traceability information. This property answers the following questions. How effectively does the traceability method provide the required information? How usable is it? How beneficial is it?

3.3 Affordability

The problem of high costs of traceability is one of the hindrances to industry adoption [54]. Affordability is a property that is concerned with minimizing costs. Cost can be measured in terms of level of effort needed to train users, level of manual intervention required, and level of effort needed to develop custom code to enable traceability across different tools.

3.4 Maintainability

This property is concerned with the ease of maintaining traceability links. One of the main challenges with adopting a traceability method is the high rate of link deterioration. Software artifacts are constantly modified but the links are not updated.

3.5 Accessibility

Accessibility refers to the ability of various stakeholders to navigate to the traced artifacts. There are three dimensions in this property: heterogeneity of artifacts, heterogeneity of tools, and the distribution of artifacts across different groups in a development team. It is a challenge to trace between heterogeneous artifacts since they are at different levels of formality. It is equally challenging to trace artifacts across heterogeneous tools due to lack of tool interoperability [55, 59]. Finally, tracing artifacts across different groups is a challenge due to the lack of communication between groups [68]. Understanding mechanisms on how to effectively share artifacts used by other groups is essential.

3.6 Scalability

Due to the numerous artifacts produced in a software lifecycle, it is important for a traceability approach to be scalable. Scalability is a property that addresses the ability to trace artifacts in large-scale projects. Scalability can be evaluated in terms of the amount of storage needed for trace information, and number of users.

3.7 Customizability

Since traceability is perceived differently by different users [68], customizing a traceability approach is essential. This property addresses the ease of customizing traceability according to the following categories: domain, project, and user. Domain-specific customization is the ability to adapt a traceability approach to a specific domain [18]. Project-specific customization is the ability to adapt a traceability approach to a specific project. Different organizations have different traceability requirements which may also vary from project to project [59]. Finally, user-specific customization is the ability to adapt a traceability approach to the different needs of the various stakeholders [68]. This customization enables stakeholders to directly benefit from the trace information they capture [31].

3.8 Auditability

This property addresses the ability of third party auditing of traceability links. An outside user should be able to follow the traceability links to answer specific questions such as which requirements were tested, what is the project scope, who implemented a subsystem, what is the rationale for the design, etc.

We have just covered the required properties for effective end-to-end traceability. We now introduce the domain of e-Science in the next section.

4 Overview of e-Science

Scientists are increasingly relying on large-scale computation to perform experiments [71]. The burgeoning field of e-Science, in which computational resources are heavily used in scientific research [10, 98], involves collaborating teams of scientists who use distributed and heterogeneous resources to accomplish shared goals [70]. *In silico* experiments, which are performed on the computer or via computer simulation [49], and data analysis are conducted in scientific fields such as high energy physics [1], biology [11, 14], geosciences [13], and engineering [12]. *In silico* experiments enable further data analysis on existing data as well as the formulation of hypotheses that can be tested in the laboratory [87].

4.1 Importance of Repeatability

Despite the shift to increased usage of computational resources, the basic requirement for repeatability of experiments still holds. The results of an experiment are compromised if one is not able to identify the data source and processing applied to it. In some cases, the acceptance of results by the community hangs on the ability to reproduce the experiment [60]. Another reason for the importance of repeatability is the need to verify other researcher's results and further one's own research [37]. Tracking contextual information, such as users of the experiment, the rationale behind the experiment, and details about the experiment run, is essential to the scientific process [104].

Due to the increased complexity of experiments and increased size of data sets, sometimes in the range of petabytes [1, 40], it is now more important to record the execution of an experiment because re-running experiments is a time-consuming effort. Thus, the automated capture of data provenance, the series of transformations applied to input data [56], has become necessary for repeatability of experiments. Consequently, provenance support for *in silico* experiments has recently received wide attention in the literature in different scientific domains [104]. Grand Challenges have been issued [5] and several workshops [15-17, 19] have been organized to increase discussion on the topic of data provenance. Numerous techniques [24, 33, 42, 50, 56, 85] have been proposed and new tools [26, 36, 40, 46, 57, 58, 67, 71, 104, 114] have been developed to address data provenance.

The main approach to capturing data provenance is through the use of scientific workflows [104] since they not only make it feasible to capture the individual workflow run, but they also codify the design of the experiment or scientific analysis. Introduced more than a decade ago [101,

109], scientific workflows have gained wide adoption today among scientists due to their capabilities of tracking data and ordered transformations on data [104]. There are also other approaches that enable the capture of data provenance, such as the automated capture of user interaction with data [40] and the capture of file-related events [46].

4.2 Definitions

Definition: e-Science is defined as "[s]cience increasingly performed through distributed global collaborations enabled by the Internet, using very large data collections, terascale computing resources and high performance visualizations" [98].

Definition: Data provenance defined as the “origin and history of data” [104], enables the identification of the series of transformations applied to an input data [56]. Data provenance is also defined as the “metadata about experiment processes, the derivation paths of data, and the sources and quality of experimental components, which includes the scientists themselves, related literature, etc” [113]. Data provenance enables repeatability of experiments, verification and reproduction of data, and validation of *in silico* experiments that cannot be checked against the real world [71]. Other terms used for data provenance are data lineage or data pedigree, depending on the type of processing applied to data [38].

Definition: Scientific workflow is defined as “a series of structured activities and computations that arise in scientific problem-solving” [101], and an “automated process that combines data and processes in a structured set of steps to implement computational solutions to a scientific problem” [26]. Scientific workflows are distinguished from business workflows in that scientific workflows are data-centric [56, 82, 109], are more flexible [97, 109], and are mainly used for running experiments [97].

We have briefly looked at e-Science and introduced terms that will be used for the remainder of the survey. The next section juxtaposes the areas of data provenance in e-Science with software traceability in software engineering. Interestingly, the problems in e-Science can also be analyzed from the economic, technical and social perspectives.

5 Why Data Provenance Potentially Provides Insights to the Software Traceability Problem

This section explains why insights can potentially be gained from data provenance approaches. Many similarities exist between these two domains, in terms of benefits, challenges, and requirements. Like a software product, a scientific workflow is an intellectual product that is subject to intellectual property [79]. In addition, distributed collaborative research teams in e-Science are similar to distributed software project teams in software development. Table 2 below summarizes other similarities.

	Data Provenance	Software Traceability
Terminology	provenance, lineage, data pedigree	traces
Item traced	data sets	software artifacts
Context	experiment run	software project
	organization	organization
	domain representation	domain of application
		laws and regulations

Table 2: Similarities Between Data Provenance and Software Traceability

5.1 Similar Benefits

The benefits of capturing data provenance parallel the benefits of establishing traces between artifacts. One benefit to provenance is the ability to understand the significance of experimental results. Similarly, traceability enables a more comprehensive understanding of the software product being developed. Another benefit is the ability to assess the impact of a change. Data provenance enables users to understand which processes need to be re-run due to parameter changes. Software traceability also enables users to identify the dependent artifacts that must be changed. A third benefit is the enhanced communication between scientists. Data provenance records enable other scientists to understand the methods of analysis used in an experiment. Similarly, traceability allows software engineers to understand the rationale behind an artifact by being able to follow traces to related documents. Still another benefit to data provenance is the ability to verify experiments. Similarly, traceability also validates that requirements have been met. Other benefits include the enabling of third party auditing, the lowering of cost through re-use, and the identifying of bugs in the system.

5.2 Similar Challenges

Many of the manifestations of the software traceability problem parallel the problems in e-Science. In fact, the problems in e-Science can also be classified into the three perspectives we mentioned: economic, technical, and social (see Table 3).

	Data Provenance	Software Traceability
Economic	Overhead in provenance capture (e.g. manual annotations, managing resources, metadata storage) [104]	Overhead in establishing and maintaining traceability links [77, 92]
Technical	Incompatibility of independent components [70].	Inability to trace artifacts across independent tools used due to lack of interoperability [59, 92].
Technical	Complex semantic links between data sources [24]	Implicit and complex relationships between artifacts [27]
Technical	Different data formats (input/output formats) [104]	Different levels of abstraction [81]
Social	Difficulty in sharing information across different researchers (different perspectives) [70]	Different perspectives of various stakeholders in a software development team on what to trace [68]

Table 3: Similar Challenges Between Data Provenance and Software Traceability

5.3 Similar Requirements

Although there are different uses for provenance across the different fields in science, the basic requirements of recording, querying, and processing provenance information are applicable across science [84]. These parallel the requirements in software traceability for defining traceability links (recording provenance) and utilizing trace links (querying and processing provenance). Other requirements in data provenance such as enabling the reuse of the experiment's process, summarizing experimental results for project management, and enabling other scientists to use the information [84, 104] are also similar to the software traceability requirements of enabling reuse of software artifacts, collecting of progress statistics, and enabling the accessibility of information across different stakeholders. Finally, the requirement for detailed, accurate, and reproducible audits of experiments [104] parallels the requirements for third party auditing in a software development setting.

5.4 Comparing and Contrasting Lifecycles

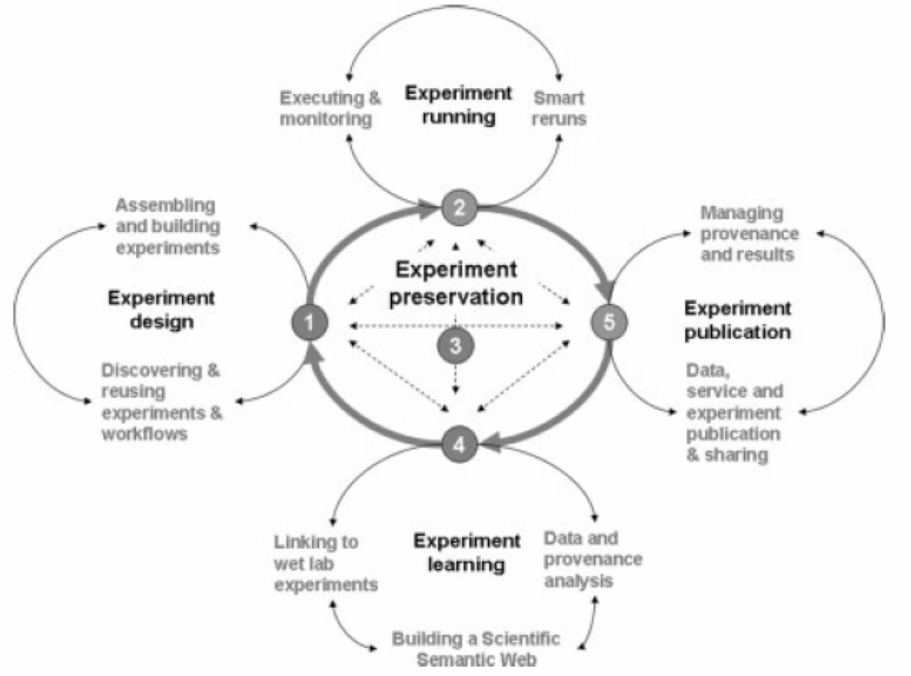

Figure 2: Life Cycle of *In Silico* Experiment [104]

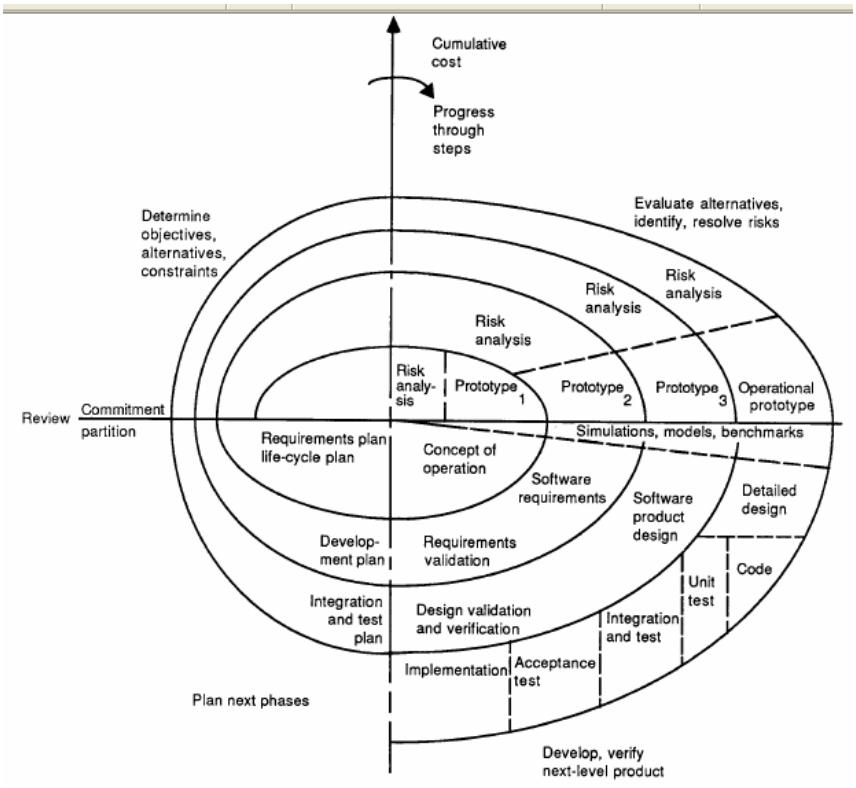

Figure 3: Software Lifecycle - Spiral Model [35]

The *in silico* experiment lifecycle is very similar to the software development lifecycle. The stages of experiment design, experiment running, experiment publication [104] (see Figure 2) are analogous to the software design, implementation, and deployment phases. However, experiment preservation and experiment learning in the experiment life cycle [104] do not have counterparts in the software lifecycle. It can be argued that risk analysis phase in the Spiral Development Life Cycle [35] (see Figure 3) has some semblance with the idea of learning from past experience. Finally, the requirements elicitation phase in software development is not present in an *in silico* experiment lifecycle since the end-user and the developer of the experiment are the same person: the scientist. Thus, the experimental design can be considered "correct" for an entire experiment lifecycle. The design is then modified in subsequent lifecycles to test other hypotheses.

5.5 Differences

There are also some differences between data provenance and software traceability. In data provenance, data products are more constrained (i.e. restricted to one level of abstraction) while software artifacts exist at different levels of formality. However, even though data provenance is restricted to one level of formality, it still has problems with incompatible formats, incompatible data types, and incompatible semantic domains [33]. Thus, data integration techniques that address these problems could possibly be applied to software traceability. In addition, data products in themselves do not carry any semantic information without added user annotation. Software artifacts, on the other hand, carry some semantic information that is understandable to humans. Even in the presence of this semantic information, metadata such as timestamps and authorship would help to inform how an artifact relates to other artifacts. Techniques that link data products with metadata would thus potentially be useful in software traceability.

Lastly, in data provenance systems like scientific workflows, all processes are automated. In other words, all manipulations to a data product are performed by a computational object. Consequently, the data product is amenable to automatic provenance capture. Meanwhile in software development, processes are mainly performed by humans and capturing traces automatically may be more difficult. However, even with automated provenance capture, user annotations are still needed to supplement the captured data provenance. Thus, insights from both automatic and manual provenance capture will still be useful in capturing traces.

5.6 Discussion

Due to the similarities between data provenance and software traceability, we conjecture that data provenance techniques could improve software traceability. Since scientific workflows have led the way in providing provenance support [14, 105], most of the tools surveyed come from this category. Since the recording of process and decisions [90] along with software artifacts is important in software traceability, we believe that insights from scientific workflows will especially be useful because scientific workflows raise the visibility of the process of manipulating and transforming data [88]. It is also important to note that scientific workflow systems, like Pegasus [59] and Condor [58], are not surveyed because they do not emphasize provenance capture but focus on scheduling jobs on the grid and providing reliable access to

high performance computing. In addition, provenance systems that focus on fine-grained provenance [48] such as scientific databases are also not surveyed. While these techniques are useful in closed world domains, artifacts in a software development setting are not constrained to transformations between databases. Yet, there are data integration techniques from scientific databases that may be useful, and they are briefly mentioned in Section 7.6.1.1. Other tools and techniques surveyed are selected based on the data provenance insights they provide.

We discussed the similarities between software traceability and data provenance. We also explained why we can potentially glean insights from data provenance. The next section presents related surveys in the areas of data provenance and software traceability.

6 Related Surveys

While surveys in the topics of software traceability and data provenance in e-Science already exist, this survey is unique in that it seeks insights from data provenance to apply to the software traceability problem. In this section, we give an overview of related surveys in both software traceability and data provenance.

An extensive survey on software traceability [103] discusses the state of the art in the field and covers topics such as trace link representation, the different uses of traceability, and approaches for generating and maintaining trace relations. Interestingly, the survey attributes the traceability problem to issues reflecting the three perspectives we identified in Section 2.4. Meanwhile, a survey of tools and approaches that aim to tackle the requirements traceability problem is provided in [69]. Since the focus of that work is identifying the sources of requirements, difficulties are attributed to human and organizational problems and high start-up costs.

In the area of e-Science, data provenance systems are categorized according to usage, type of provenance (i.e. data-oriented or process-oriented), representation, storage, and method of dissemination [100]. Data provenance systems are also categorized according to their architectures (e.g. service-oriented, database, command processing, scripting) [100]. Another means of classifying of provenance systems is along the key functions in data lineage [37]. Meanwhile, scientific processing systems are evaluated according to their workflow model, metadata model, and capabilities for lineage retrieval [38]. A taxonomy of scientific workflow systems is presented in [112]. Scientific workflows are classified according to workflow specification, workflow execution, error handling, information retrieval, and movement of data.

We have just covered related surveys in software traceability and data provenance. The next section surveys data provenance systems. Specific data provenance techniques are also discussed where appropriate.

7 How Data Provenance Systems/Techniques Potentially Meet Software Traceability Requirements

In this section, we survey various data provenance systems. We then analyze the ways that these systems potentially meet the software traceability requirements.

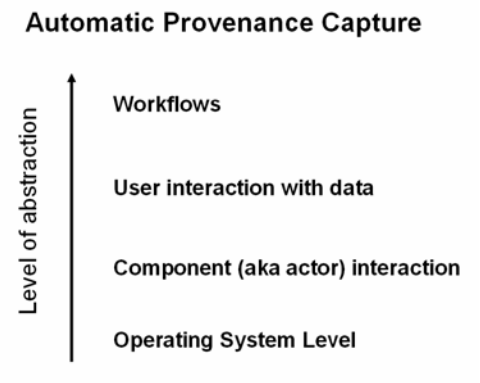

Figure 4: Categories of Automatic Provenance Capture

7.1 System Overviews

This section provides an overview of the provenance systems that we survey. The systems are categorized by the level of automatic provenance capture they provide (see Figure 4).

7.1.1 Category: Workflows

This category of provenance systems is characterized by awareness of process. The steps in an experiment or data analysis are manually encoded in the workflow specification. The workflow specification is then used as the basis for executing the experiment or analysis. Similar to the User Interaction category, provenance capture is coarse-grained.

Kepler

Kepler, a workflow editing environment based on Ptolemy II, enables actor-oriented modeling [82]. An actor is a component that represents a data or computational object [25]. A workflow consists of both actors and ports, the communication channels between actors. Actors pass tokens that contain information through ports [44]. The director oversees the communication between actors and the overall workflow coordination [43, 82]. Kepler records provenance by capturing events at the communication ports. Used in the domains of ecology, geology, biology, astrophysics, and chemistry [25], Kepler is an open-source project [4] with contributors from various projects including the Science Environment for Ecological Knowledge (SEEK) [8] and the Cyber infrastructure for the Geosciences (GEON) [13]. Although Kepler is still at its beta version, it has an active user mailing list [4] and is currently used by the German Research Center for Artificial Intelligence [108] to process spatial information.

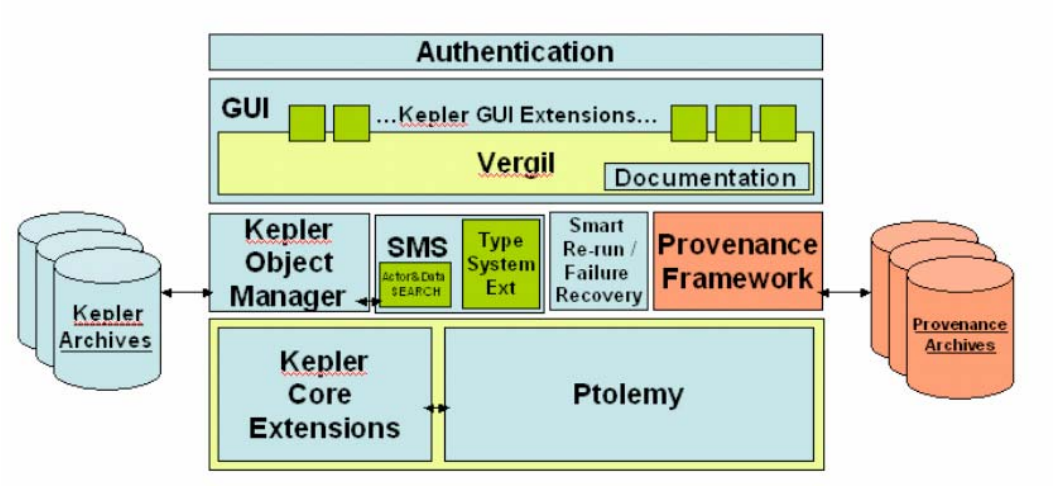

Figure 7: Kepler System Architecture [26]

Taverna

Taverna is part of the myGrid project which is geared towards supporting the rapid prototyping of *in silico* experiments in the domain of bioinformatics [104]. A characteristic of the bioinformatics domain is the emphasis on the exchange of data analysis rather than the execution of computationally intensive experiments on the grid [87]. myGrid presents the Provenance Pyramid (see Figure 8) which classifies provenance into the following levels: process level, data level, organization level, and knowledge level [104]. A provenance service in the myGrid project, COHSE (Conceptual Open Hypermedia Services Environment) [113], dynamically generates hyperlinks between provenance logs and documents sets based on the attached semantic concepts from the myGrid ontology. Taverna uses the Resource Description Framework (RDF) to represent provenance. As an open source project, Taverna has gone through several official releases [3]. Taverna has been successfully used in the genetic research of the Graves Disease and Williams-Beuren Syndrome and it is now being used in various biological research projects [3].

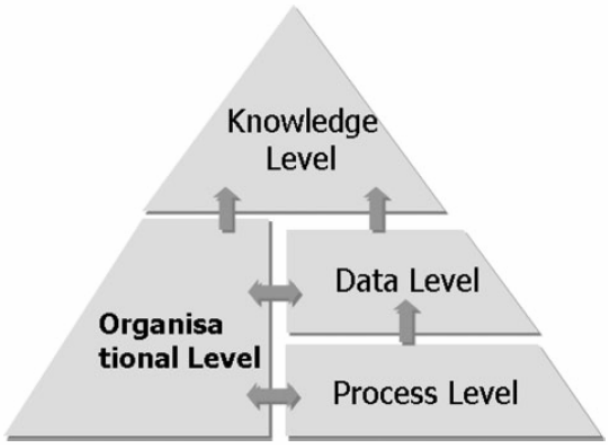

Figure 8: Provenance Pyramid [104]

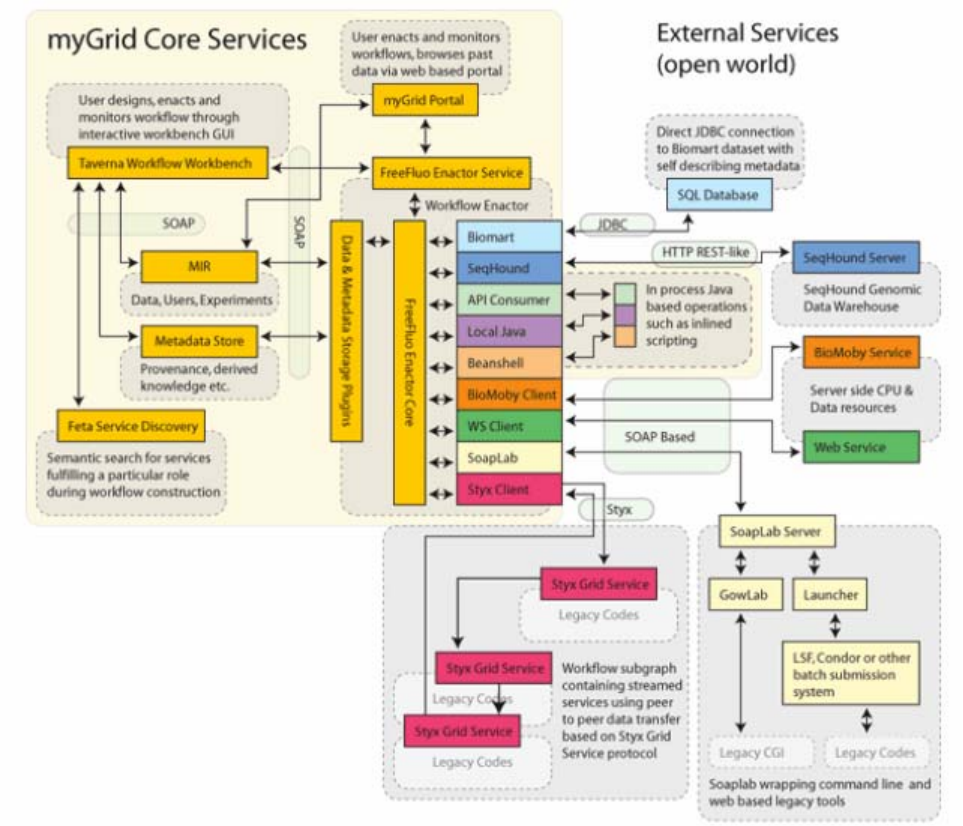

Figure 9: Taverna Architecture and myGrid Components [3]

VisTrails

VisTrails tracks and records steps performed by the user in a scientific exploratory process [67]. It captures changes to both a workflow instance (e.g. parameter value change) and a workflow specification (e.g. modules and connection change). VisTrails is the first system to enable capture of workflow evolution, and it uses an action-based mechanism to capture both the provenance of data products and workflows. In a VisTrails tree, a node corresponds to a version of a workflow. Nodes in the VisTrails version tree are never deleted; thus, the infrastructure is similar to a version control system. The VisTrails Builder is the workflow editor and the VisTrails Repository stores workflow specifications. Previously saved workflows can be retrieved through the VisTrails Server or imported into the Visualization Spreadsheet where they can also be modified. The workflow is executed by the VisTrails Cache Execution Manager. VisTrails is an open source tool [99] that is currently in beta version [7]. VisTrails is currently being used in radiation oncology and environmental observation and forecasting [7].

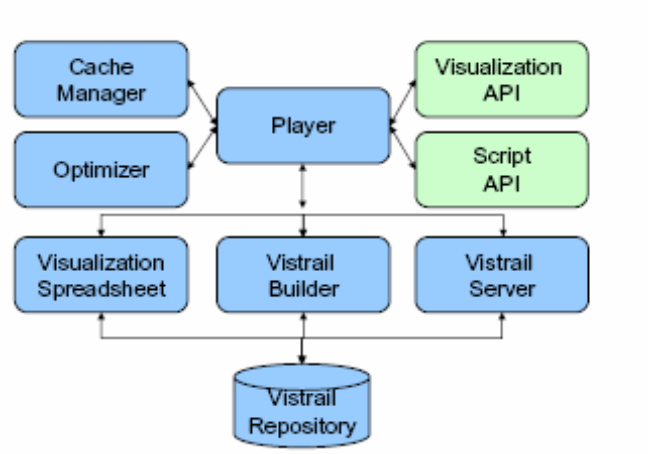

Figure 10: VisTrails Architecture [67]

Virtual Data System

The Virtual Data System (VDS) is a workflow system that enables the maintenance and accessibility of distributed and decentralized provenance information. Resources are maintained by local groups and federated indexes are used to enable access to provenance information owned by other groups [114]. VDS treats all resources, i.e. data and computational objects, as first class entities. Thus, all resources are assigned types and are represented in a schema (see Figure 11). The term "virtual data" is used to represent data that may not yet exist but can be defined by the transformations that will be applied to it. A Virtual Data Catalog stores the information defined by the schema. A part of the GriPhyN project [115], VDS has been applied to the domains of high energy physics, astronomy, and neuroscience [66, 114]. VDS has several official releases [6], and VDS is currently being used in projects such as ATLAS (high energy physics event simulator), FOAM (ocean and atmospheric modeling), and GADU (Genomics) [6].

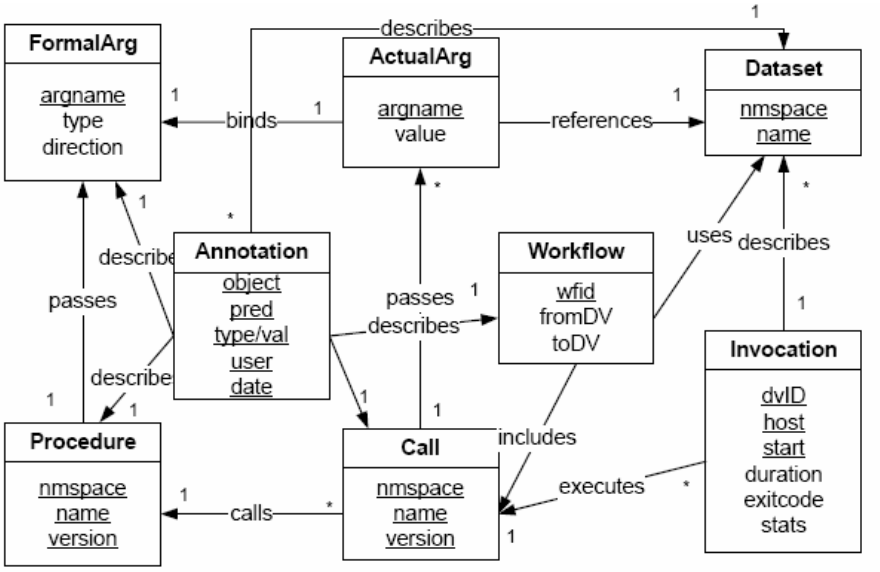

Figure 11: Virtual Data Schema [114]

7.1.2 Category: User Interaction with Data

This category of provenance systems automatically captures the user interactions with data. No prior specification of the processing steps is needed in this coarse-grained provenance capture.

CAVES

The Collaborative Analysis Versioning Environment System (CAVES) project supports the collaboration between researchers or groups of researchers in the domain of high energy physics by enabling the capture of scientific analyses [40]. Users select which analyses will be published to a remote server and made available to other researchers and which analyses will be stored in their local machines. CAVES extends ROOT, an object-oriented data analysis framework that is widely used in high energy physics. CAVES uses CVS and plug-in extensions for ROOT to support provenance capture. CAVES records the series of data manipulation commands issued by the user and saves them into a log in the repository. Provenance logs can be extracted from the repository to reproduce a virtual data product on demand. (Virtual data are data products that are either produced in the past or may be generated in the future based on a well defined method of production). CAVES is designed to support a large community of users running large amounts of simulated and real data on the scale of tens and hundreds of petabytes. One specific application of this project is to support scientists who are running experiments at the Large Hadron Collider at CERN in Geneva, Switzerland. CAVES has also been used to capture data analysis as demonstrated at Supercomputing 2005 [41]. An official version of CAVES is available for download [41].

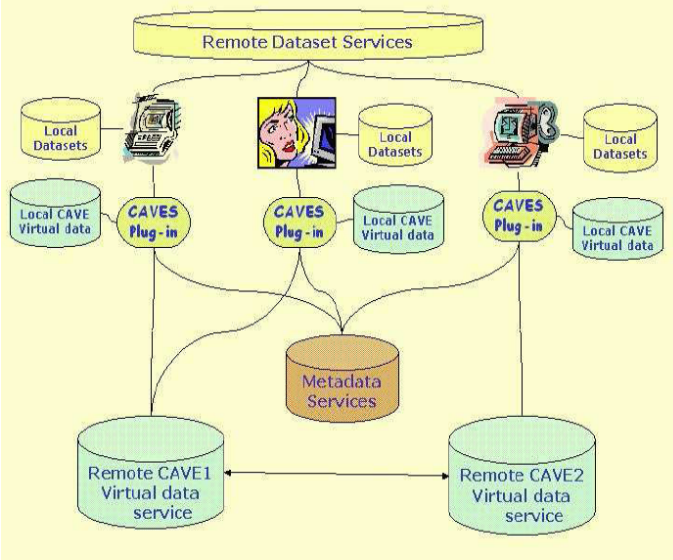

Figure 5: CAVES Distributed Repositories [40]

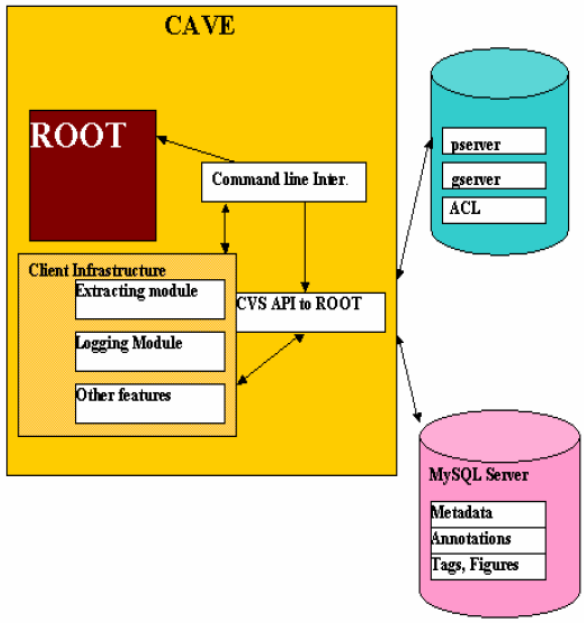

Figure 6: CAVES Architecture [39]

7.1.3 Category: Component (a.k.a. Actor) Interaction

In this category of provenance systems, the basis for automatic provenance capture is the interactions between components. Thus, there is no awareness of the process.

PreServ

PreServ, an implementation of the Provenance Recording Protocol (PReP), is designed to enable provenance capture among heterogeneous actors. An actor is either a client or a service in the Service Oriented Architecture. This approach enables two types of provenance capture: interactions between actors (referred to as interaction p-assertion) and internal actor state (referred to as actor state p-assertion). Provenance capture is accomplished by wrapping services and capturing the interactions between services through the exchanged messages [45, 71]. Provenance can be recorded as long as each service implement the protocol, i.e. the APIs, specified by PReP [71]. A part of the PASOA (Provenance Aware Service Oriented Architecture) project, PreServ is used in the domain of bioinformatics [2, 71]. PreServ can be downloaded from [2].

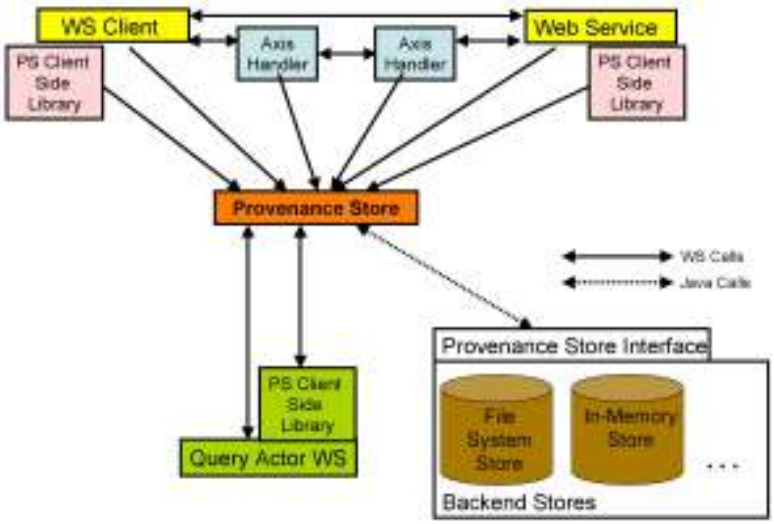

Figure 12: PreServ Component Interactions [2]

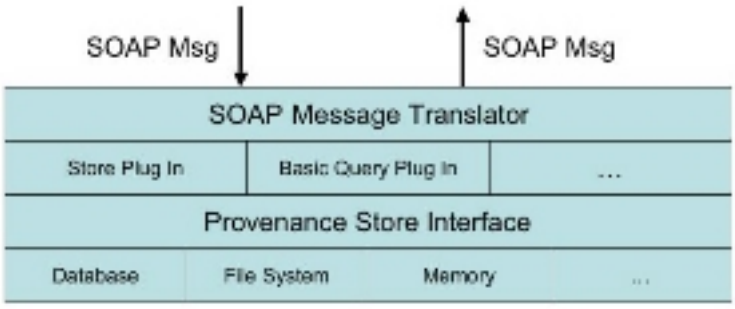

Figure 13: Layered Design of the Provenance Store [2]

7.1.4 Category: Operating System Level

This category of automatic provenance capture has no awareness of actor entities. This fine-grained provenance capture records all observed operating system level events.

PASS

Based on the capture of all observed file-related events, Provenance-Aware Storage System (PASS) enables automatic provenance collection across heterogeneous tools without requiring the user to manually enter information and without using applications that explicitly collect provenance information. Observed provenance collection is the approach PASS uses for recording provenance. The information collected is also more complete since it explicitly records environment information such as configuration, environment variables, etc. A drawback to this approach is it can lead to "false provenance" or extraneous provenance. Other difficulties with observed provenance collection, such as mismatch in granularity, large numbers of versions, and introduction of cycles, are addressed by PASS. Since provenance is captured at a much lower granularity than expected by the user, PASS supports manual user annotations to allow the grouping of provenance into coarser granularities. To address difficulties with versioning, PASS supports "copy-on-write" versioning, in which a new version is created for every first file-write event (between the file-open and file-close events). Cycles in provenance are also checked every time an edge is added. Observed provenance collection can be used to supplement the disclosed provenance approach, i.e. provenance collection is specified by users or tools [46]. PASS can be obtained by contacting the research group [9].

We have just introduced relevant provenance systems. We now evaluate these tools according to the software traceability properties identified in Section 3.

7.2 Capturability

The following subsections discuss the automated and manual capture of provenance data. In each category, the kind of provenance information captured, the capture methods, and the granularity of capture are discussed. It is interesting to note that most of the tools surveyed [26, 40, 67, 104, 114] supplement automatically captured provenance with manual user annotations. Section 7.2.3 further discusses the types of semantics provided by these different tools.

7.2.1 Automated Capture

Various provenance systems automatically capture provenance at different levels of abstraction (see Figure 4). These systems range from capturing all low level operating system processes without awareness of provenance, as in PASS [46], to high level capture of researcher interaction with the data, as in CAVES [40]. Meanwhile, in scientific workflows systems like Kepler, Taverna, and VDS, the user specifies the series of transformations that will be applied on the data through a workflow specification language [26, 104, 114]. During the workflow execution, provenance is captured in the form of events [26], service invocations [104], or general invocations (e.g. executable programs, SQL command queries, or method invocations) [65]. In these systems, input and output data are associated with the captured provenance information. Another system in the scientific workflow category, VisTrails, captures the workflow evolution

itself [67]. Thus, VisTrails supports tracking changes to a workflow instance (e.g. changes to parameter values) and to a workflow specification (e.g. changes to modules and connections). At the component interaction level, PreServ automatically captures the series of interactions between services as well as the internal states of the services [71].

There are also different methods of automated capture. Across these different methods of capture, chronological ordering is preserved. CAVES extends a popular data analysis framework, ROOT, to automatically log all changes to the data [40]. Meanwhile, Kepler uses a Provenance Recorder to capture registered events at the communication ports between actors [26]. Taverna's workflow enactment engine stores the order of services invoked in a process log [104]. VisTrails records action-based provenance such as the user interaction with the workflow [67]. VDS, on the other hand, wraps a workflow execution with a parent process which uses operating system events to capture the execution [114]. PreServ records the interaction between services [71]. Finally, PASS observes all processes that run on a PASS enabled operating system and captures all operating system events associated with a data file [46].

The granularity of automated capture varies among the different tools surveyed. Capture at the granularity of user-interaction is performed by CAVES and VisTrails [40, 67]. Tools such as Taverna and VDS automatically capture provenance at the granularity of service invocation [104, 114]. Meanwhile, PreServ captures at the granularity of service interaction and internal service state [71]. Capture at the granularity of events are performed by Kepler and PASS [26, 46]. Kepler captures at the granularity of component (a.k.a. actor) events [26] while PASS captures at the granularity of operating system events [46].

7.2.2 Manual Capture

Most of the tools surveyed enable manual capture of provenance data to supplement the automatically captured provenance. In scientific workflows like Kepler, Taverna, VisTrails, and VDS [26, 67, 104], users manually specify the sequence of processing as a workflow. The workflow specification is then executed after physical resources are mapped to the logical representations. Besides the workflow specification, Kepler and Taverna enable the manual capture of context by having users specify the context of the experiment (e.g. who, what, where, when, and why) in Kepler [26] and organizational information (e.g. user, creator, organization, project, hypothesis, experiment design) in Taverna [104]. Taverna also enables users to enter knowledge level information such as user notes and domain-specific information. In VisTrails, users may also tag selected versions of the workflow with a name [67]. Finally, CAVES and PASS allow free form annotations [40, 46].

There are different methods for manually entering provenance information. CAVES provide an 'annotate' command that enables users to attach annotations to a uniquely identified dataset [40]. Names or tags may also be attached to important workflow versions in VisTrails [67]. Meanwhile, formal annotations are specified by Taverna and VDS. Taverna supports the use of ontology in the myGrid registry to annotate the workflow [113]. Finally, annotations in VDS should conform to the Virtual Data Schema [114].

The granularity of manual capture varies by tool. Annotations may be at the workflow level as in Kepler and Taverna [26, 104] or at a specific workflow version as in VisTrails [67]. Annotations may also be at the data product level as in CAVES [40] or at the file level as in PASS [46]. Finally, VDS enables annotations at the level of virtual data objects (i.e. procedure, parameters, workflows, data sets) as defined by the schema [114].

7.2.3 Capture Trace Semantics

In scientific computing, achieving reproducibility and repeatability requires a high level of exactness to be recorded: the right input dataset, the correct version of a component, the correct series of transformation, the infrastructure support (i.e. libraries or modules used), and the hardware used. At the same time, all resources, data and computational objects, must be given semantic annotations, i.e. their real world representation and how they relate to each other. Even though software traceability does not require the same level of exactness for reproducibility, the techniques in capturing or inferring semantics may be applicable to software traceability.

7.2.3.1 The Importance of Context

In e-science, context aids in evaluating the results of an experiment or scientific analysis [104]. Context can be broadly defined as "anything that was true" during an experiment run [84]. There are two levels of context [104]. First, context refers to the details of the experiment run, i.e. provenance [104]. Context can also refer to information surrounding experiments, including hypotheses, conclusions, findings, users, creators, organizations, projects, personal notes, domain-specific information, and time and location of the experiment run [26, 87, 104].

7.2.3.1.1 Context: Experiment Run

In an experiment run, contextual information can identify the types of relationships between objects. These relationships can be classified by dependency, time, or contributor. Dependency relationships between different versions of a workflow are captured in VisTrails [67]. Since a version tree represents the workflow evolution, a child node is dependent on the parent nodes. Another example of a dependency relationship is the relationship of objects with the execution environment since reproducing same results may depend on the kernel module loaded, libraries installed, etc [46]. Temporal relationship are another type of relationship provided by context. An example is the chronological order of services invoked by a workflow engine. The order of invocation is automatically captured by workflow systems [104]. The chronological order of workflow versions created are also captured in VisTrails and are represented by the color of the nodes [67]. Contributor relationships identify which entities initiate the production of a data object. Contributors can be users [40] or services [71].

Meanwhile, specific relationship types between objects can also be specified. VDS captures semantic relationships between objects by assigning types to all scientific resources [114]. The relationships are represented by the virtual data schema.

7.2.3.1.2 Context: Information Surrounding an Experiment

Information surrounding an experiment is equally important in evaluating the results of an experiment. The provenance pyramid (see Figure 8) illustrates the relationship between context captured as a workflow run, i.e. data and process provenance, and context as information surrounding an experiment [104]. The Provenance pyramid classifies the surrounding

information of an experiment as organizational (e.g. user, creator, hypothesis, project, organization, experiment design) and knowledge-based (e.g. user notes, domain relationships). Context information also include time and location of an experiment run and the rationale behind the experiment [26].

7.2.3.2 Categories of Captured Semantics in the Literature

7.2.3.2.1 Prospective vs. Retrospective Provenance

This category distinguishes between a planned experiment or analysis (a.k.a. prospective) and a record of an experiment or analysis already performed (a.k.a. retrospective) [38]. Prospective provenance is represented as a workflow specification [114]. Meanwhile retrospective provenance is represented by the workflow execution or the data lineage which contains mappings to the physical resources used, e.g. data sets, functions, etc. [38, 114]. It is useful to combine the information from both types of provenance. Prospective provenance aids in understanding retrospective provenance since it is specified at a higher level of abstraction. Meanwhile, retrospective provenance fills in the missing details of prospective provenance.

7.2.3.2.2 Observed Actor Interaction vs. Internal Actor Provenance

Another distinction is the source of provenance information. Observed actor interaction is provenance recorded by other services that interact with an actor. Meanwhile, internal actor provenance is provenance recorded by an actor about its own internal events [72]. Observed actor interaction provenance is useful in cross checking the recorded provenance among multiple sources.

7.2.3.2.3 Observed vs. Disclosed Provenance

This distinction identifies provenance collection based on what has been determined *a priori* (disclosed) and provenance collection based on recording all events (observed). Disclosed provenance provides more semantics, but is limited to collecting provenance within the provenance system. Observed provenance does not have this limitation. However, collected provenance does not have semantics and could even contain “false provenance” [46].

7.2.3.2.4 Internal vs. External Provenance

This category is concerned with provenance collected within a provenance system and provenance collected outside a provenance system. One technique to achieve external provenance is database integration. The ability to obtain external provenance is difficult and may sometimes entail manual processes, such as the manual curation of databases [48].

7.2.3.2.5 Logical vs. Infrastructural Provenance

Provenance information can be distinguished between logical and infrastructural concerns. Logical provenance refers to the data and the transformations on the data while infrastructural provenance refers to the environment on which the transformations are performed, such as the environment variables and the state of hardware [94]. This is an important distinction since provenance recording should not be limited to capturing the processes surrounding the data. The assumptions about the environment should also be recorded.

7.2.3.3 Inferring Semantics

There are a couple of ways that semantic relationships between objects can be inferred. One way is through the use of a reasoner such as the Description Logic reasoner which automatically classifies concepts into hierarchies [113]. Since these concepts are attached to provenance logs and document sets, provenance logs can be related to publications or experiment notes that are of the same concepts in the hierarchy. Another way to infer semantic relationships is through the use of an inference engine. New relationships are inferred between indirectly related data products by applying backward and forward chaining to a knowledge base. The knowledge base consists of captured provenance represented as an RDF and inference rules [50].

7.2.4 Discussion

We note that the level of automatic semantic capture increases with the increased awareness of semantics in the framework of the provenance tool. For example, since the CAVES project is based on an existing data analysis tool [40], it is easy to infer the semantics of a provenance log. Similarly, the data flow is specified in workflows [43]. Thus, the ordering of processes invoked reflects the ordering of transformations applied to a given dataset. In contrast, PASS has no awareness of semantics and thus will capture all observed events related to a file [46].

Meanwhile, the increased built-in semantics in the framework limits the ability to capture provenance information from different tools outside the framework (known as level of openness). For example, CAVES is only able to track provenance within the data analysis framework [40] while PASS enables provenance tracking across different tools as long as the host's operating system is PASS enabled [46]. Thus, there is a tradeoff between the ability to automatically capture the semantics of provenance information and the ability to capture provenance across heterogeneous tools.

7.3 Affordability

The following subsections discuss the costs involved in using data provenance tools. Cost is measured in terms of training time to use a provenance tool, effort in manual provenance capture, and effort in developing custom code to capture provenance across heterogeneous tools.

7.3.1 Training Time to Use a Provenance Tool

The cost of training time varies between the different tools. Since CAVES builds upon an existing data analysis tool [40], the cost associated with training existing users is low to none. The cost of training for PASS [46] is also low to none since most of the provenance capture is automatic. Kepler [43] and Taverna [104] have a low cost since the graphical workflow language used is easy to learn. VisTrails [67] also has a low cost since it is easy to learn to navigate through the recorded workflow space.

7.3.2 Effort in Manual Provenance Capture

The cost of manual provenance capture varies from low to high between tools. VisTrails [67] has low overhead since users simply assign names to noteworthy versions of the workflow. CAVES [40] and PASS [46] also have low cost since the manual annotations are based on what users deem important. VDS [114] has a low to medium cost in provenance capture since

annotations must comply with the schema. Taverna [104] has a high cost in manual provenance capture. Capturing knowledge level annotations require users to relate the experiment with domain-specific concepts through the use of myGrid ontology. The high cost provides benefits that are not realized in other tools. Mapping experiments to ontology enables the dynamic generation of hyperlinks between provenance information and other document sets.

7.3.3 Effort in Developing Custom Code

The effort needed to develop code extensions that capture of provenance across different tools also varies. PASS [46] has virtually no cost since it is able to capture all the system level calls to any tool or function as long as the operating system is PASS-enabled. PreServ [71] has a low cost since it only requires custom wrappers on existing systems to intercept provenance information. The other workflow systems like Kepler, Taverna and VDS require that the components conform to their framework. Meanwhile, CAVES [40] has no allowance for integrating other tools since provenance capture is limited to the ROOT data analysis tool.

7.4 Utilizability

Provenance information is used to reproduce data [40, 104]. Provenance information is also used to identify problems with workflow execution [26] as well as to identify differences with the execution environment [46]. Saved workflows are reused [26, 67] to save time in re-running the analysis. Ontology-related provenance is also used to dynamically generate hypertext [113]. Other uses of trace information include browsing [40, 67, 104], querying [71, 104, 114], and comparing workflow versions (VisTrails) [67].

7.5 Maintainability

These data provenance systems do not have many explicit techniques for achieving maintainability. VDS [114] enables object updates through the use of SQL. CAVES [40] enables browsing virtual data products and annotations, which users can retrieve and update.

7.6 Accessibility

The following subsections discuss the ways that provenance systems enable the accessibility of provenance data. Accessibility is examined along the following dimensions: heterogeneous data, heterogeneous tools, and different groups.

7.6.1 Heterogeneous Data

In e-Science, the issue of heterogeneous data is generally addressed when integrating data from different sources and when composing heterogeneous components into a workflow. The following subsections discuss the techniques used in handling heterogeneous data.

7.6.1.1 Data Integration

The problem of data integration is especially prevalent in the life sciences domain since it is difficult to determine whether records from different databases represent the same data. This problem has come to be known as the “Life Science Identity Crisis” [104]. The myGrid project and Taverna address this problem by uniquely identifying data with a life science identification

(LSID). However, the problem still exists when integrating with data from outside myGrid. Another technique used in data integration is manually extracting data from various sources and producing a manually curated database [48]. There is a high cost to this approach, but the database is considered high quality by the community of users.

7.6.1.2 Input/Output Data Mismatch

When creating the workflow out of different components, the problem of matching the required input of one component with the given output of another component exists [104]. Taverna addresses this problem through the use of "shim services", which reconcile differences between the input and output data [104]. Another approach is through the use of ontology. Different ontologies are used to classify mismatches. Parameters are annotated with concepts from the ontology. When there is an identified mismatch, the library of transformations based on mismatch type is checked and the appropriate conversion is used [33]. In a similar approach, the system automatically checks for the data and semantic type compatibility when users specify the workflow [24]. An ontology is used to determine semantic compatibility. When a mismatch is detected, the tool automatically inserts a conversion rule. Generating an automatic conversion program is also used in resolving incompatible formats between semantically compatible input and output data [42]. Another approach used in Kepler is annotating parameters to enable semi-automated transformations [88].

7.6.2 Heterogeneous Tools

One means of enabling provenance capture among heterogeneous components is through the use of event listeners and message passing, as in Taverna [104]. A plug-in listens to subscribed events and generates messages once it receives an event. Another method of capturing provenance among different tools is by wrapping code around these tools so that their interaction with the provenance store is via a specified API, as in PreServ [71]. Lastly, systems like PASS [46] capture provenance over heterogeneous tools by intercepting system level calls. In such systems, it is not necessary to develop custom code for different tools.

7.6.3 Different Groups

The surveyed tools present several techniques for improving accessibility of provenance information across distributed groups. One way is to enable users to publish their provenance information in a remote repository that is accessible to other users as in CAVES and VisTrails [40, 67]. VisTrails has the added functionality of enabling scientists to synchronize their changes to the workflow [67]. Meanwhile, VDS enables distributed groups to access provenance information owned by other groups through the use of federated indexes [114]. Users query the inter-catalog references which then point to locally owned Virtual Data Catalogs.

7.7 Scalability

There are various approaches to achieving scalability. Since captured provenance information can quickly become large, scalability in storage is important. CAVES, VDS, PreServ enable the use of distributed repositories to store provenance information [40, 71, 114]. Distributed and decentralized VDS repositories are owned by local groups [114]. PreServ also enables the use of multiple types of repositories as long as the interaction is through the PReP API [71]. Other

techniques in minimizing storage overhead include pruning selected provenance information, compressing provenance information, combining frequently accessed attributes, and deleting irrelevant attributes [46].

These tools can also scale to many users. CAVES and VDS have no upper limit to the number of users [40, 114].

7.8 Customizability

The following subsections discuss the extent of customization provided by the provenance systems. The three types of customization examined are domain-specific, project-specific, and user-specific customization.

7.8.1 Domain-Specific Customization

Among the tools surveyed, only Taverna enables domain specific customization to the provenance information captured [104]. Users enter annotations that map experiments to domain-specific concepts.

7.8.2 Project-Specific Customization

Project-specific customization is achieved in a couple of ways. Virtual organizations may adopt naming conventions in assigning unique identification to their virtual data products, as in CAVES [40]. In VDS, virtual data objects (i.e. data and components) are maintained in a distributed and decentralized context. Thus, local groups can choose the virtual data objects to store in their local VDS [66]. Furthermore, groups may also maintain "overlay" information, or a separate set of annotations, on objects owned by other groups.

7.8.3 User-Specific Customization

There are different means for achieving user-specific customization. Users have control over the time at which to capture provenance information [71] and the level of granularity at which to capture provenance [26]. Users also have control over which provenance to publish [40] and which information to keep in their local machine or workspace [40, 114]. Users may also maintain their own custom metadata over the same set of data sets or virtual data objects [104, 114]. In Taverna, users may use their custom ontology to create different views of the provenance [104].

There are also other techniques for enabling user-customized views. The provenance information presented to a user can be customized with user views [56] and scoped provenance queries [85]. Scoped provenance queries enable users to limit provenance information by filtering out a type of relationship, internal operations of an actor, or a data role. Another means of customizing provenance views is by limiting access to provenance information based on access policies [50].

7.9 Auditability

A third party entity may audit a scientific analysis or experiment mainly by reproducing the data set through the use of captured provenance as in Taverna [104] and VDS [65]. CAVES also enables on-demand reproducibility of data by extracting the derivation log from the server [40].

One of the challenges to auditability is that different results could be produced even though the same input data and the same workflow is executed. The difference lies in the execution environment. PASS [46] addresses this challenge by capturing details about the execution environment, such as the kernel modules used and the libraries installed.

We have just surveyed the ways in which data provenance systems and techniques meet the requirements for end-to-end software traceability. The next section highlights the lessons learned from the survey.

8 What are the Lessons Learned?

This section highlights the insights gained from the survey of data provenance techniques. These insights will inform an approach to tackle the software traceability problem.

Insight: ***Use shared semantic concepts to automatically generate trace links***
Ontology is used to dynamically generate hypertext linking provenance logs with various documents to form a web of science [113]. Since one of the shortcomings of automatic trace recovery is the lack of link semantics, using domain-specific vocabularies or ontologies to improve the automatic generation of trace links is a possible solution [103].

Insight: ***Provenance of data is intertwined with provenance of process***
Provenance of data can be viewed as the process that brought the data to its current state [47, 85]. In addition, the provenance of data identifies which processes were used to reach a conclusion or an output [107]. Thus, tracing the process enables the tracing of data. Scientific workflow systems are especially suited to track processes [104]. In software traceability, it is also important to trace both product and process objects [77]. Documenting the development processes aids in selecting which artifacts to trace. PRO-ART is a software traceability system that records both the artifacts and the processes that manipulate the artifacts (e.g. generating, removing, editing artifacts) [89]. However, there is a high upfront cost in recording processes since all products and interrelationships must first be formally represented in a schema. Using provenance capture techniques can potentially lower the cost of recording the processes that manipulate the artifacts

Insight: ***Context can inform the type of relationship between artifacts***
The two types of contextual information in e-Science are the provenance information captured during an experiment run and the information surrounding an experiment. The first type aids in understanding how data is manipulated while the second type describes how data is related to entities in the real world. Context in an experiment run frames assumptions about the

environment (e.g. hardware environment, loaded libraries, means of communication between independent computational objects) and how entities are related (e.g. "input data X and intermediate data Y are fed into computation object A to produce output data Z"). In addition, context as information surrounding an experiment links the objects in an experiment run to what they represent in the real world. In the domain of software engineering, context is also important. Context can inform the assumptions made when artifacts were generated (e.g. company conventions, regulatory requirement, time restrictions) and how artifacts are related (e.g. "Use Cases are developed before the system is implemented, and QA engineers develop test suites using the Functional Requirements Document and Design document, but they do not look at the code").

Insight: ***The type of provenance information captured is directly related to provenance usage***
The type of provenance information captured depends on how and where it will be used [48]. *How* provenance information will be used is demonstrated in the use of workflows. Workflows [26, 67, 104] are used to encapsulate a scientific analysis or experiment by explicitly specifying the series of transformations that will be applied to an input data. *Where* provenance information will be used is also important. CAVES tracks user interaction with the data within the ROOT data analysis tool for several reasons. It minimizes the training time for users since the intended users already use the tool. Recording the scientific analysis within ROOT also enables the scientific analysis to be reproduced on demand within ROOT. In this situation, enabling provenance to be captured across heterogeneous tools is unnecessary.

Insight: ***Reasoners help in automatically inferring relationships***
One of the capabilities afforded by some provenance systems is the ability to reason, i.e. analyze, query, and browse captured provenance [45, 91]. Reasoners can also be applied to infer semantic relationships between objects [50, 113]. For instance, the Description Logic reasoner classifies concepts into a hierarchy. Consequently provenance logs with annotated concepts can be related to other provenance logs or documentation with concepts in the same hierarchy. The reasoner used in [45] can sometimes infer the data provenance even with missing provenance information. An inference engine can be used to infer new relationships between indirectly related objects [50].

Insight: ***Automated provenance capture has limits***
It is interesting to note that most of the tools surveyed [26, 40, 46, 67, 104, 114] supplement the automated provenance capture with manual capture, e.g. user annotation. This indicates that regardless of the approach, automated techniques are limited in the scope of information they can capture. Automatic capture is limited to events observable by a computer (e.g. user issued commands [40], communication between computational objects [26, 71], and file events on the operating system level [46]). The rationale, purpose, and other extrinsic information cannot be automatically captured and must be manually entered by the user.

Insight: ***Cost in collecting data provenance is invested into the scientific process***
Scientists have traditionally maintained experiment records in their log books [104]. Thus, for scientists, the cost of record keeping is invested into the scientific process. Without a scientist's log of past experiments, it would be difficult for a scientist to analyze the results. It would also be equally difficult to remember specific details about the experiment. Scientists are willing to manually specify the design of the workflow since it will be the basis for running their

experiment while at the same time serving as a future reference for further experiments [26, 36, 104]. In this scenario, the producer of data provenance is the same as the consumer of data provenance.

Insight: ***Cost in collecting data provenance is outweighed by the utility of the information***
Manually curated databases are integrated from different data sources by scientists. This effort usually involves examining publications and going over databases. Even though this process is costly in terms of labor hours, it is outweighed by the perceived benefit since manually curated databases are considered "higher quality" by the community [48]. In this scenario, even though the producers are different from consumers, there is a well-understood benefit to their activity (i.e. benefit to the community of users). It is also important to note that individuals who collect provenance information have a good understanding of the data that should be included [48].

Insight: ***There is a tradeoff between cost and the level of manual semantic capture***
There is a tradeoff between minimizing cost and increasing the level of semantic understanding among the manual captured techniques. Lightweight approaches [40, 46, 67] have lower cost but capture less semantics. Meanwhile, annotating provenance logs with concepts from the ontology is a heavyweight approach with high cost. However, these annotations enable the dynamic generation of hyperlinks between provenance logs and publications with similar concepts [113]. This link between provenance and relevant publications provides a better understanding of the data.

Insight: ***There is a tradeoff between the level of "openness" and the level of automatic semantic capture***
"Openness" is the ability to capture provenance among heterogeneous tools [71]. The higher the level of openness, the more capable it is in capturing provenance across heterogeneous tools. However, the more open is the provenance capture, the lower the level of semantics that can be captured. For example, CAVES [40] has no means of capturing provenance with other tools, but the provenance captured has a high level of semantics since it is directly related to the data analysis tool. On the other end of the spectrum, PASS [46] has a high level of openness, i.e. can capture provenance across any tool, but the level of semantics is none.

Insight: ***Provenance collection is stakeholder-centric***
Users control provenance collection. Users determine what to record [26, 40], when to record [26, 71], where to store provenance information [40, 66], what to publish [40], and what level of granularity to record [26]. Users may also provide information to enable user-customized views on provenance [104, 114]. The individual who usually collects provenance information is the same individual who uses that provenance information [56, 84]. We learn from software traceability case studies like [32][92] that stakeholder role is directly associated with the expected usage of trace information. For instance, a system designer is interested in using a trace tool to record the design rationale and to understand the impact of a requirements change on system modules and test plans [92]. Meanwhile, a QA engineer is interested in using a trace tool to ensure that test plans cover all the requirements [92]. A traceability approach should cater to these varied stakeholder interests to provide direct benefits to them.

Insight: ***Provenance systems enable local ownership and global access***
The concept of local ownership and global access is provided by provenance systems like CAVES [40], Taverna [104], and VDS [114]. Local users or groups have complete control over their provenance information. At the same time, provenance information is visible to external groups via globally accessible servers as in CAVES [40] or federated indexes as in VDS [114]. Taverna also enables groups to maintain annotations on provenance data owned by other groups [104].

9 Conclusion: How to Apply Insights to Software Traceability?

Software traceability, despite its recognized importance in software development, has largely been unachievable in practice. We analyzed the reported problems with implementing software traceability from the economic, technical, and social perspectives. These perspectives are intertwined and must be tackled simultaneously. Then we examined how similar problems are solved in e-Science, a domain with similar characteristics to software engineering. Data provenance systems and techniques in e-Science were surveyed to gain potential insights in approaching the software traceability problem. We now conclude with the application of these insights to software traceability.

Addressing the economic perspective
Cost is a major inhibiting factor to industry adoption of a traceability approach [32]. By integrating traceability with existing development processes, the cost of traceability can be considered part of the overall development costs. In addition, by understanding how traceability information can benefit software development, e.g. by significantly lowering software maintenance cost, the benefits can outweigh the cost of tracing.

Addressing the technical perspective
The technical perspective is concerned with addressing the explosion of the artifacts space, maintenance of trace links, heterogeneity of artifacts, and heterogeneity of tools. An insight from e-Science is that data provenance collection should be directly related to provenance usage. This insight can be used to address the explosion of the artifact space. The types of artifacts, the types of relationships and the granularity at which they are traced should reflect the expected usage of the trace information. Case studies like [92] provide some examples of expected uses of trace information. To address trace link maintenance, reasoning techniques such as bulk query and bulk updates [114] can be used to keep the links updated. To address the heterogeneity of artifacts, using a transformation program [42, 104] to automatically convert between artifact types perhaps provide a solution. However, this is limited in cases where the information can easily be mapped from one artifact type to another. To address the heterogeneity of tools, events or protocols can be used to automatically cascade additions or changes of artifacts across different tools. This way, redundant data entry and manual reconciliation of data can be avoided. This approach is limited to those artifacts that are at the same level of formality and abstraction.

Addressing the social perspective
The social perspective is concerned with addressing the distribution of artifacts across different groups, the different user expectations of a traceability tool, and the low motivation for performing traceability tasks. To address tracing across different groups, one can publish

artifacts publicly or use global pointers [40, 114]. Groups may publish artifacts to a server accessible by other groups [40]. Similar to the federated index used in VDS, a global pointer directs users to the traced artifacts owned by different groups [114]. To address different expectations of the tool, users should have the option of customizing the traceability tool. For example, users should have the option to maintain their own trace information, i.e. similar to custom ontology [104] or "overlay" information [114], over the set of traced artifacts. Automatically generated customized views of traceability will help provide user-specific customization. To address the problem of low motivation, there should be a clear benefit to the task of traceability. There is a distinct difference in the level of motivation between scientists and software engineers. Scientists are willing to keep records while most software engineers dislike traceability tasks. Scientists invest their time in collecting provenance because they directly benefit from the provenance information, i.e. they are the consumers of information [104]. Therefore, enabling software engineers to directly benefit from traceability will potentially encourage them to perform traceability tasks. Even in cases where the producers are not necessarily the consumers of data, such as in the case of manually curated databases [48], high value could still be assigned to the task of tracing the source of data. Thus, assigning a high value to traced information can potentially motivate software engineers to perform traceability tasks.

Addressing the economic, technical, and social perspectives

To minimize cost, to enable the automatic capture of semantics, and to increase the quality of links as determined by human analysts, a combination of automated and manual approaches should be used. First, an automatic capture of the context and the processes that generate and manipulate artifacts enables the automatic capture of trace semantics, e.g. dependency, temporal, and contributor relationships. Then, lightweight annotations, can be attached to an artifact by the user who generated or manipulated the artifact. This procedure enables the dynamic generation of semantically-rich traceability links at lower costs. The quality of links increases since knowledgeable users attach the annotations to traced artifacts.

While the possible approaches presented in this survey are by no means complete, these provide a good start for exploring the solution space of the end-to-end software traceability problem. These approaches are well-grounded since they have already been applied in the similar field of e-Science. We hope that a feasible traceability approach that is adoptable in industry will result from this endeavor.

10 References

[1] *Grid Physics Network (GriPhyN)*. <http://www.griphyn.org/>.

[2] *Provenance Aware Service Oriented Architecture*. <http://www.pasoa.org/>.

[3] *myGrid*. <http://www.mygrid.org.uk/>.

[4] *Kepler Project*. <http://www.kepler-project.org/>.

[5] *Provenance Challenge Wiki*. <http://twiki.ipaw.info/bin/view/Challenge/>.

[6] *VDS - The GriPhyN Virtual Data System* <http://www.ci.uchicago.edu/wiki/bin/view/VDS/VDSWeb/WebMain>.

[7] *VisTrails Wiki*. <http://www.vistrails.org/>.

[8] *Science Environment for Ecological Knowledge (SEEK)*. <http://seek.ecoinformatics.org/>.

[9] *PASS: Provenance-Aware Storage Systems*. <http://www.eecs.harvard.edu/~syrah/pass/>.

[10] *2nd IEEE International Conference on e-Science and Grid Computing*. <http://www.escience-meeting.org/>.

[11] *E-Science Resource for High Throughput Protein Crystallography (e-HTPX Project)*. <http://clyde.dl.ac.uk/e-htpx/index.htm>.

[12] *Grid enabled Optimisation and Design Search for Engineering (Geodise)*. <http://www.geodise.org/>.

[13] *Geosciences Network (GEON)*. <http://www.geongrid.org/>.

[14] *Genomics: GTL - Systems Biology for Energy and Environment*. <http://genomicsgtl.energy.gov/>.

[15] *Workshop on Data Derivation and Provenance*. <http://www-fp.mcs.anl.gov/~foster/provenance/>, 2002.

[16] *Data Provenance and Annotation Workshop*. <http://www.nesc.ac.uk/esi/events/304/>, 2003.

[17] *International Provenance and Annotation Workshop (IPAW)*. <http://www.ipaw.info/ipaw06/>, 2006.

[18] *3rd ECMDA Traceability Workshop*. <http://modelbased.net/ecmda-traceability/>, 2007.

[19] *Principles of Provenance Workshop*. <http://www.cis.upenn.edu/~plclub/propr/>, 2007.

[20] Aizenbud-Reshef, N., Nolan, B.T., et al. Model Traceability. *IBM Systems Journal.* 45(3), p. 515-26, 2006.

[21] Alexander, I. Towards Automatic Traceability in Industrial Practice. In *Proc. of the 1st International Workshop on Traceability.* p. 26-31, 2002.

[22] Alford, M. A Requirements Engineering Methodology for Real-time Processing Requirements. In *Proc. of the 2nd International Conference on Software Engineering.* San Francisco, California, US, 1976.

[23] Almeida, J., van Eck, P., et al. Requirements Traceability and Transformation Conformance in Model-Driven Development. In *Proc. of the 10th IEEE International Enterprise Distributed Object Computing Conference.* Hong Kong, China, Oct 16-20 2006.

[24] Altintas, I., Bhagwanani, S., et al. A Modeling and Execution Environment for Distributed Scientific Workflows. In *Proc. of the 15th International Conference on Scientific and Statistical Database Management.* Cambridge, MA, July 9-11, 2003.

[25] Altintas, I., Berkley, C., et al. Kepler: An Extensible System for Design and Execution of Scientific Workflows. In *Proc. of the 16th International Conference on Scientific and Statistical Database Management.* Santorini Island, Greece, June 21-23 2004.

[26] Altintas, I., Barney, O., et al. Provenance Collection Support in the Kepler Scientific Workflow System. In *International Provenance and Annotation Workshop (IPAW)*: Chicago, IL, 2006.

[27] Anderson, K.M., Sherba, S.A., et al. Towards Large-Scale Information Integration. In *Proc. of the 24th Intl. Conference on Software Engineering.* Orlando, Florida, May, 2002.

[28] Antoniol, G., Caprile, B., et al. Design-code Traceability Recovery: Selecting the Basic Linkage Properties. *Elsevier. Science of Computer Programming.* 40(2-3), p. 213-34, July, 2001.

[29] Antoniol, G., Canfora, G., et al. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering.* 28(10), p. 970-83, Oct, 2002.

[30] Appleton, B. *The Trouble with Tracing: Traceability Dissected*. <http://www.cmcrossroads.com/articles/agile-cm-environments/the-trouble-with-tracing%3a-traceability-dissected.html>, CM Crossroads, 2005.

[31] Arkley, P. and Riddle, S. Overcoming the Traceability Benefit Problem. In *Proc. of the 13th IEEE International Conference on Requirements Engineering.* Paris, France, Aug 29 - Sep 2, 2005.

[32] Asuncion, H., François, F., et al. An End-To-End Industrial Software Traceability Tool. In *Proc. of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Int'l Symposium on the Foundations of Software Engineering (ESEC/FSE).* Dubrovnik,Croatia, September, 2007.

[33] Belhajjame, K., Embury, S.M., et al. On Characterising and Identifying Mismatches in Scientific Workflows. In *3rd International Workshop on Data Integration in the Life Sciences (DILS)*: Hinxton, UK, 2006.

[34] Bianchi, A., Fasolino Anna, R., et al. An Exploratory Case Study of the Maintenance Effectiveness of Traceability Models. In *Proc. of the 8th International Workshop on Program Comprehension (IWPC).* Limerick, Ireland. , June 10-11, 2000.

[35] Boehm, B.W. A Spiral Model of Software Development and Enhancement. *IEEE Computer*. 21(5), p. 61-72, May, 1988.

[36] Boose, E.R., Ellison, A.M., et al. Ensuring Reliable Datasets for Environmental Models and Forecasts. In *Proc. of the 5th International Conference on Ecological Informatics.* Santa Barbara, CA, Dec 4-7, 2006.

[37] Bose, R. A Conceptual Framework for Composing and Managing Scientific Data Lineage. In *Proc. of the 4th International Conference on Scientific and Statistical Database Management.* Edinburgh, UK, July 24-26 2002.

[38] Bose, R. and Frew, J. Lineage Retrieval for Scientific Data Processing: A Survey. *ACM Computing Surveys.* 37(1), p. 1-28, March, 2005.

[39] Bourilkov, D. *CAVES/CODESH*. <http://bourilko.web.cern.ch/bourilko/caves.html>.

[40] Bourilkov, D. THE CAVES Project - Collaborative Analysis Versioning Environment System. *International Journal of Modern Physics.* 20, p. 3889-3892, 2005.

[41] Bourilkov, D., Khandelwal, V., et al. Virtual Logbooks and Collaboration in Science and Software Development. In *Proc. of the International Provenance and Annotation Workshop (IPAW).* Chicago, IL, May 3-5, 2006.

[42] Bowers, S. and Ludasher, B. An Ontology-Driven Framework for Data Transformation in Scientific Workflows. In *First International Workshop on Data Integration in the Life Sciences (DILS)*: Leipzig, Germany, 2004.

[43] Bowers, S. and Ludascher, B. Actor-Oriented Design of Scientific Workflows. In *Proc. of the 24th International Conference on Conceptual Modeling.* Klagenfurt, Austria, Oct 24-28, 2005.

[44] Bowers, S., McPhillips, T., et al. A Model for User-Oriented Data Provenance in Pipelined Scientific Workflows. In *International Provenance and Annotation Workshop (IPAW)* 2006.

[45] Branco, M. and Moreau, L. Enabling Provenance on Large Scale e-Science Applications. In *Proc. of the International Provenance and Annotation Workshop (IPAW).* Chicago, IL, May 3-5, 2006.

[46] Braun, U., Garfinkel, S., et al. Issues in Automatic Provenance Collection. In *International Provenance and Annotation Workshop (IPAW)*: Chicago, IL, 2006.

[47] Buneman, P., Khanna, S., et al. Why and Where: A Characterization of Data Provenance. In *Proc. of the 8th International Conference on Database Theory*. London, UK, Jan 4-6, 2001.

[48] Buneman, P., Chapman, A., et al. A Provenance Model for Manually Curated Data. In *Proc. of the International Provenance and Annotation Workshop (IPAW).* Chicago, IL, May 3-5, 2006.

[49] Cavalcanti, M.C., Targino, R., et al. Managing structural genomic workflows using Web services. *Data & Knowledge Engineering.* 53(1), p. 45-74, 2005.

[50] Cheung, K. and Hunter, J. Provenance Explorer - Customized Provenance Views Using Semantic Inferencing. In *Proc. of the 5th International Semantic Web Conference.* Athens, GA, Nov 5-9, 2006.

[51] Cleland-Huang, J., Chang, C.K., et al. Event-based Traceability for Managing Evolutionary Change. *IEEE Transactions on Software Engineering*. 29(9), p. 796-810, Sep, 2003.

[52] Cleland-Huang, J., Zemont, G., et al. A Heterogeneous Solution for Improving the Return on Investment of Requirements Traceability. In *Proc. of the 12th IEEE International Requirements Engineering Conference.* p. 230-239, Kyoto, Japan, Sept 6-11, 2004.

[53] Cleland-Huang, J. Toward Improved Traceability of Non-Functional Requirements. In *3rd International Workshop on Traceability in Emerging Forms of Software Engineering*. p. 14-19 ACM Press: Long Beach, CA, 2005.

[54] Cleland-Huang, J. Just Enough Requirements Traceability. In *Proc. of the 30th Annual International Computer Software and Applications Conference (COMPSAC).* Chicago, IL, Sep 17-21, 2006.

[55] Cleland-Huang, J., Berenbach, B., et al. Best Practices for Automated Traceability. *Computer*. 40(6), p. 27-35, 2007.

[56] Cohen, S., Cohen-Boulakia, S., et al. Towards a Model of Provenance and User Views in Scientific Workflows. In *Proc. of the Data Integration in the Life Sciences. Third International Workshop, DILS 2006.* Hinxton, UK, July 20-22, 2006.

[57] Couvares, P., Kosar, T., et al. Workflow Management in Condor. In *Workflows for e-Science*, Taylor, I.J., et al. eds. p. 357-375, Springer: London, 2007.

[58] Deelman, E., Mehta, G., et al. Pegasus: Mapping Large-Scale Workflows to Distributed Resources. In *Workflows for e-Science*, Taylor, I.J., et al. eds. p. 376-393, Springer: London, 2007.

[59] Domges, R. and Pohl, K. Adapting Traceability Environments to Project Specific Needs. *Communications of the ACM*. 41(12), p. 54-62, 1998.

[60] Dvorak, F., Koufil, D., et al. gLite Job Provenance. In *International Provenance and Annotation Workshop (IPAW).* Moreau, L. and Foster, I., Editors: Chicago, IL, 2006.
[61] Egyed, A. A Scenario-driven Approach to Traceability. In *Proc. of the 23rd International Conference on Software Engineering.* p. 123-132, Toronto, Ontario, Canada, May 12-19, 2001.
[62] Egyed, A., Biffl, S., et al. A Value-based Approach for Understanding Cost-benefit Trade-offs During Automated Software Traceability. In *3rd International Workshop on Traceability in Emerging Forms of Software Engineering*. p. 2-7, ACM Press: Long Beach, CA, 2005.
[63] Egyed, A., Biffl, S., et al. Determining the Cost-Quality Trade-Off for Automated Software Traceability. In *20th IEEE/ACM international Conference on Automated Software Engineering*ACM Press: Long Beach, CA, USA, 2005.
[64] Evans, M. SPMN Director Identifies 16 Critical Software Practices *CrossTalk, The Journal of Defense Software Engineering.* March, 2001.
[65] Foster, I., Vockler, J., et al. Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. In *Proc. of the 14th International Conference on Scientific and Statistical Database Management.* p. July 24-26 Edinburgh, UK, 2002.
[66] Foster, I., Vockler, J., et al. The Virtual Data Grid: A New Model and Architecture for Data-Intensive Collaboration. In *Proc. of the 15th International Conference on Scientific and Statistical Database Management.* Cambridge, MA, July 9-11, 2003.
[67] Freire, J., Silva, C.T., et al. Managing Rapidly-Evolving Scientific Workflows. In *Proc. of the International Provenance and Annotation Workshop (IPAW).* Chicago, IL, 3-5 May, 2006.
[68] Gotel, O. and Finkelstein, C. An Analysis of the Requirements Traceability Problem. In *Proc. of the 1st Intl. Conference on Requirements Engineering.* p. 94-101, Los Alamitos, CA, 1994.
[69] Gotel, O. *Contribution Structures for Requirements Engineering.* Thesis (Ph.D) Thesis. Imperial College of Science, Technology, and Medicine, 1996. <http://www-dse.doc.ic.ac.uk/dse-papers/viewpoints/olly_phd_thesis.ps.Z>.
[70] Groth, P., Luck, M., et al. A Protocol for Recording Provenance in Service-Oriented Grids. In *Proc. of the 8th International Conference on Principles of Distributed Systems (OPODIS).* Grenoble, France, Dec 15-17 2004.
[71] Groth, P., Miles, S., et al. Recording and Using Provenance in a Protein Compressibility Experiment. In *Proc. of the 14th IEEE International Sympsoium on High Performance Distributed Computing.* Research Triangle Park, NC, July 24-27 2005.
[72] Groth, P., Miles, S., et al. Principles of High Quality Documentation for Provenance: a Philosophical Discussion. In *Proc. of the International Provenance and Annotation Workshop (IPAW).* Chicago, IL, May 3-5, 2006.
[73] Hamilton, V.L. and Beeby, M.L. Issues of Traceability in Integrating Tools. In *Proc. of the IEE Colloquium on Tools and Techniques for Maintaining Traceability During Design.* London, UK, Dec 2, 1991.
[74] Hayes, J. and Dekhtyar, A. *Grand Challenges for Traceability*. Center of Excellence for Traceability, Technical Report COET-GCT-06-01-0.9, 2007.
[75] Hayes, J.H., Dekhtyar, A., et al. Helping Analysts Trace Requirements: An Objective Look. In *Proc. of the 12th IEEE International Requirements Engineering Conference.* p. 249-59, Kyoto, Japan, Sept 6-11, 2004.

[76] Hayes, J.H. and Dekhtyar, A. Humans in the Traceability Loop: Can't Live with 'Em, Can't Live Without 'Em. In *Proc. of the 3rd Intl. Workshop on Traceability in Emerging Forms of Software Engineering.* p. 20-23, Long Beach, CA, Nov 8, 2005.

[77] Jarke, M. Requirements Tracing. *Communications ACM.* 41(12), p. 32-36, Dec, 1998.

[78] Jarke, M., Bui, X.T., et al. Scenario Management: An Interdisciplinary Approach. *Requirements Engineering Journal.* p. 155-173, 1998.

[79] Khan, I., Schroeter, R., et al. Implementing a Secure Annotation Service. In *Proc. of the International Provenance and Annotation Workshop (IPAW).* Chicago, IL, 3-5 May, 2006.

[80] Leffingwell, D. and Widrig, D. *The Role of Requirements Traceability in System Development*. <http://www-128.ibm.com/developerworks/rational/library/content/RationalEdge/sep02/TraceabilitySep02.pdf>, 2002.

[81] Lindvall, M. and Sandahl, K. Practical Implications of Traceability. *Software - Practice and Experience.* 26(10), p. 1161-80, 1996.

[82] Ludäscher, B., Altintas, I., et al. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice and Experience.* 18(10), p. 1039-1065, 2006.

[83] Marcus, A. and Maletic, J.I. Recovering Documentation-To-Source-Code Traceability Links Using Latent Semantic Indexing. In *Proc. of the Proceedings of the 25th International Conference on Software Engineering.* Portland, OR, 2003.

[84] Miles, S., Groth, P., et al. *The Requirements of Recording and Using Provenance in e-Science Experiments.* Electronics and Computer Science, University of Southampton, Technical Report, 2005.

[85] Miles, S. Electronically Querying for the Provenance of Entities. In *International Provenance and Annotation Workshop (IPAW).* Moreau, L. and Foster, I., Editors: Chicago, IL, 2006.

[86] Neumuller, C. and Grunbacher, P. Automating Software Traceability in Very Small Companies - a Case Study and Lessons Learned. In *Proc. of the 21st IEEE International Conference on Automated Software Engineering.* Tokyo, Japan, Sep 18-22, 2006.

[87] Oinn, T., Greenwood, M., et al. Taverna: Lessons in Creating a Workflow Environment for the Life Sciences. *Concurrency and Computation: Practice and Experience.* 18(10), p. 1067-1100, 2006.

[88] Pennington, D., Higgins, D., et al. Ecological Niche Modeling Using the Kepler Workflow System. In *Workflows for e-Science,* Taylor, I.J., et al. eds. p. 91-108, Springer: London, 2007.

[89] Pohl, K. and Jacobs, S. Concurrent Engineering: Enabling Traceability and Mutual Understanding. *Concurrent Engineering: Research and Applications.* 2(4), p. 279-90, 1994.

[90] Pohl, K., Brandenburg, M., et al. Integrating Requirement and Architecture Information: A Scenario and Meta-Model Based Approach. In *7th Intl. Workshop on Requirements Engineering: Foundation for Software Quality*, 2001.

[91] Rajbhandari, S. and Walker, D.W. Support for Provenance in a Service-based Computing Grid. In *Proc. of the e-Science All-Hands Meeting* Nottingham, UK, Aug 31-Sep 3, 2004.

[92] Ramesh, B., Powers, T., et al. Implementing Requirements Traceability: A Case Study. In *Proc. of the 1995 Intl. Symposium on Requirements Engineering (RE'95).* p. 89-95, York, UK, Mar 27-29 1995.

[93] Ramesh, B. and Jarke, M. Towards Reference Models for Requirements Traceability. *IEEE Transactions in Software Engineering, 27(1),.* 27(1), p. 58-93, 2001.

[94] Reilly, C.F. and Naughton, J.F. Exploring Provenance in a Distributed Job Execution System. In *Proc. of the International Provenance and Annotation Workshop (IPAW).* Chicago, IL, May 3-5, 2006.

[95] Richardson, J. and Green, J. Automating Traceability for Generated Software Artifacts. In *Proc. of the 19th Intl. Conference on Automated Software Engineering.* p. 24-33, Linz, Austria, Sept 20-24, 2004.

[96] Schach, S.R. *Classical & Object-Oriented Software Engineering*. Fourth ed. 616 pgs., McGraw Hill, 1999.

[97] Seffino, L.A., Bauzer Medeiros, C., et al. WOODSS - A Spatial Decision Support System Based on Workflows. *Decision Support Systems.* 27(1-2), p. 105-23, 1999.

[98] Senior, I. *e-Science Definitions*. <http://e-science.ox.ac.uk/public/general/definitions.xml>, 2002.

[99] Silva, C.T., Freire, J., et al. Provenance for Visualizations: Reproducibility and Beyond. *Computing in Science & Engineering.* 9(5), p. 82-89, Sep-Oct, 2007.

[100] Simmhan, Y.L., Plale, B., et al. A Survey of Data Provenance in e-Science. p. 31-36, ACM Press, 2005.

[101] Singh, M.P. and Vouk, M.A. Scientific Workflows: Wcientific Computing Meets Transactional Workflows. In *Proc. of the NSF Workshop on Workflow and Process Automation in Information Systems: State of the Art and Future Directions.* . Athens, GA, May 8-10, 1996.

[102] Singleton, M.E. *Automating Code and Documentation Management*. Prentice-Hall, Inc.: New Jersey, 1987.

[103] Spanoudakis, G. and Zisman, A. *Software Traceability: A Roadmap* Advances in Software Engineering and Knowdledge Engineering. Chang, S.K. ed. 3, World Scientific Publishing, 2005.

[104] Stevens, R., Zhao, J., et al. Using Provenance to Manage Knowledge of In Silico Experiments. *Briefings in Bioinformatics.* 8(3), p. 183-94, May, 2007.

[105] Stevens, R.D., Tipney, H.J., et al. Exploring Williams–Beuren Syndrome Using myGrid. *Bioinformatics.* 20(1), p. i303-10, Aug 4, 2004.

[106] Sugden, R.C. and Strens, M.R. Strategies, Tactics and Methods for Handling Change. In *IEEE Symposium and Workshop on Engineering of Computer-Based Systems*: Friedrichshafen, Germany, 1996.

[107] Szomszor, M. and Moreau, L. Recording and Reasoning Over Data Provenance in Web and Grid Services. In *Proc. of the On The Move Confederated International Conferences: CoopIS, DOA, and ODBASE.* Catania, Sicily, Italy, Nov 3-7 2003.

[108] Tuot, C.J. *Kepler: A Platform to Process Spatial Information?* <http://ctuot.twoday.net/stories/4133335/>.

[109] Wainer, J., Weske, M., et al. Scientific Workflow Systems. In *Proc. of the Proceedings of NSF Workshop on Workflow and Process Automation in Information Systems: State of the Art and Future Directions.* . Athens, GA, May 8-10, 1996.

[110] Wallace, D. and Ippolito, L. A Framework for the Development and Assurance of High Integrity Software, Commerce, U.S.D.o., NIST, 1994

[111] Weidenhaupt, K., Pohl, K., et al. Scenarios in System Development: Current Practice. *IEEE Software.* 15(2), p. 34-45, 1998.

[112] Yu, J. and Buyya, R. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Record.* 34(3), p. 44-9, 2005.

[113] Zhao, J., Goble, C., et al. Semantically Linking and Browsing Provenance Logs for E-science. In *Proc. of the 1st International Federation for Information Processing Conference.* Paris, France, June 17-19 2004.
[114] Zhao, Y., Wilde, M., et al. Applying the Virtual Data Provenance Model. In *International Provenance and Annotation Workshop (IPAW)*: Chicago, IL, 2006.
[115] Zhao, Y., Wilde, M., et al. Virtual Data Language: A Typed Workflow Notation for Diversely Structured Scientific Data. In *Workflows for e-Science*, Taylor, I.J., et al. eds. p. 258-275, Springer: London, 2007.

Appendix: Survey of Provenance Systems

			CAVES [39, 40]	Kepler [26, 43, 88]	Taverna [104, 113]	VisTrails [67]	VDS [6, 65, 66, 114]	PreServ [45, 71]	PASS [46]
Project Domain			High energy physics	Ecology, Biology, Geology, Astrophysics, Chemistry	Bioinformatics		High Energy Physics, Astronomy, Neuroscience	Bioinformatics	
Project Domain					myGrid		GriPhyN	PASOA	
Capturability	Automated Capture	What	Interaction between researchers & their data analyses	Provenance events; Workflow evolution	Process provenance (how, when, where the workflow is run, data input/output, services invoked) and data provenance	Workflow evolution; changes to workflow instance (e.g parameter value change); changes to workflow specification (e.g. changes to modules and connections, operations invoked)	Workflow execution (which includes calls to transformations, executable programs, SQL command queries, method of invocation); derived files	Service interaction (interaction p-assertion); internal service state (actor state p-assertion)	Processes at the operating system level; kernel modules loaded, installed libraries, process environment
		How	Extend data analysis framework to automatically log all changes to the data. Different versions of data product and code checked into the CVS	Provenance Recorder captures registered events at the communication ports between actors and saves them to the provenance store. Provenance info is associated with a data product	Workflow enactment engine records the order of services invoked in a log. Data provenance is inferred from the process log.	Action-based provenace - records user interaction with a workflow. This is recorded as a tree with each node representing a version of a workflow	Use a parent process wrapper to capture the workflow execution and the derived files. Captured provenance is sent back to the workflow enactment engine and saved to the virtual data catalog as invocation records	Record interaction between services	Observes processes and captures low-level details
		Granularity	User-interaction with data stored by session. Ability to select between partial or complete logs	Event level	Service invocation	User-interaction with the workflow	Service invocation; dataset	Service interaction; internal service state	OS Events; Record provenance on a per-file basis
	Manual Capture	What	Free form annotation	Context (who, what, where, when, why); experiment steps or process of data analysis	Organizational (user, creator, organization, project, hypothesis, experiment design); Knowledge (personal notes, domain-specific info); experiment steps or process of data analysis	Identify selected versions; experiment steps or process of data analysis	Semantic annotation on procedures, arguments, datasets, and workflows; experiment steps or process of data analysis		Free form annotation
		How	Annotate a uniquely identified data product or component (processing unit) using the 'annotate' command in the user interface	Specify context & workflow in a workflow editor	Knowledge - use ontologies in the myGrid registry to annotate logs; specify workflow in editor	Assigning names (aka tags) to versions deemed important; specify workflow in editor	Metadata annotation defined by the virtual data schema; specify workflow in editor		Add annotation through a query tool (Provenance Explorer)
		Granularity	Data product level	Process level	Process level	Workflow version	Virtual data objects defined by the schema		File-level

Appendix: Survey of Provenance Systems

			CAVES [39, 40]	Kepler [26, 43, 88]	Taverna [104, 113]	VisTrails [67]	VDS [6, 65, 66, 114]	PreServ [45, 71]	PASS [46]
Utilizability			Provenance logs can be extracted, inspected, and modified for further analysis. Extracted provenance logs can be used to reproduce virtual data product on demand	Analyzing process provenace aids in debugging a workflow execution. Smart re-run saves time in re-running experiments where only a parameter change is involved	Use ontology to dynamically generate hypertext. Ability to query, browse provenance logs. Reproducibility	View history of workflow changes, re-use, compare workflows	Query against virtual data relationships (i.e. workflow design and execution), metadata annotation, lineage info (e.g. derived datasets, ancestor datasets)	Reasoning over provenance logs aids in determining the provenance of a dataset. Query against resulting metadata catalog	In combination with disclosed provenance techniques (e.g. scientific workflows), provenance info can aid in identifying differences in execution environment
Affordability	Training Time to Use Provenance Tool		Low to none for existing users of the ROOT data analysis tool	Low - easy to learn graphical workflow language	Low - easy to learn graphical workflow language	Low - easy to navigate through graphical workflow space	Medium - medium effort to learn scripting workflow language		Low to none. Minimal user interaction
	Manual Provenance Collection		Low - Manual annotation based on what users deem important	Low - context	High - Knowledge level annotations require high effort from experts	Low to none - assign names to versions	Low to medium. Annotation must comply with the schema		Low - Manual annotation based on what users deem important
	Developing Custom Code		No allowance for provenance collection outside the analysis tool	Provenance capture only possible with actors used in the workflow. Can wrap web services as actors	Provenance capture only possible with myGrid services must be used within the workflow		Provenance capture possible with components that conform to the Globus toolkit	Develop wrappers on existing systems to capture provenance	No custom code necessary to capture provenance. Only requirement is that OS is PASS-enabled
Maintainability			Browse data/components to retrieve and update		Not specified, although it states that provenance is collected/used at each stage in the lifecycle		Update objects using SQL	Provenance data can be queried	
Accessibility	Heterogeneous Data			Annotate parameters to partially automate transformation of output/input data	Use unique IDs (Life Science Identifier) to uniquely identify data within myGrid. External identity is still a problem. Use shim services to reconcile type mismatches between input and output data		Ability to access multiple sources of data through inter-catalog references		
	Heterogeneous Tools			Wrap webservices as actors	Integration between workflow execution engine and different components - based on plug-ins that listen to events generated.			Any component as long as it conforms to the PReP protocol of interacting with the provenance store	Capture OS level calls - automatically collect provenance on any tool invoked
	Different Groups		Users in the access control list can access analysis published in the server			Users commit selected changes. Scientists exchange patches and synchronize their VisTrails	Virtual Data Catalogs (VDCs) are maintained by local groups. Use federated indexes to access distributed VDCs		

Appendix: Survey of Provenance Systems

			CAVES [39, 40]	Kepler [26, 43, 88]	Taverna [104, 113]	VisTrails [67]	VDS [6, 65, 66, 114]	PreServ [45, 71]	PASS [46]
Scalability	Storage		Repositories, known as caves, are distributed, mirrored and synchronized. Users may store partial or complete provenance locally or remotely in a virtual data logbook				A potential limit to scalability in the method of creating federated indexes	Potentially no limit to the provenance data to be recorded since different data stores can be used as long as the PReP API is used. Different services can interact with the provenance store asynchronously.	Strategies to minimize storage overhead include pruning, merging similar information, deleting irrelevant attributes
	Number of Users		No upper limit to the number of users				No upper limit to the number of users		
Customizability	Domain Customization		None. Tool is focused on high energy physics		Manually entered annotation may contain domain-specific concepts				
	Project Customization		Virtual organizations can adopt naming conventions to label their virtual data products				Local control over what objects are stored in the VDC. May also maintain "overlay" information on objects maintained by other groups		
	User Customization		Control over which analysis history to publish and what to store in local machine	Control over what level of granularity to capture provenance. Option to save all provenance info or to recreate the data	Different users may maintain different ontologies over the same datasets. This affords different views on provenance logs		May make a permanent copy of derivation logs in their workspace. May also maintain "overlay" information on objects maintained by other groups	May choose when documentation of process should be recorded	
Auditability			Reproduce virtual data product by extracting logs from the server		Reproducibility through the use of captured provenance		(Re)derivation of data		