Grand Challenges, Benchmarks, and TraceLab: Developing Infrastructure for the Software Traceability Research Community

J. Cleland-Huang[1], A. Czauderna[1], A. Dekhtyar[2], O. Gotel, J. Huffman Hayes[3]
E. Keenan[1], G. Leach[1], J. Maletic[4], D. Poshyvanyk[5], Y. Shin[1], A. Zisman[6],
G.Antoniol[7], B. Berenbach[8], A. Egyed[9], P.Maeder[9]
Center of Excellence for Software Traceability
DePaul Univ.[1], Cal Poly[2], Univ. of Kentucky[3], Kent State Univ.[4], College of William and Mary[5]
City College, London[6], École Poly. Montréal, Siemens Corp.[8], Linz Univ.[9]
jhuang@cs.depaul.edu

ABSTRACT

The challenges of implementing successful and cost-effective traceability have created a compelling research agenda that has addressed a broad range of traceability related issues, ranging from qualitative studies of traceability users in industry to very technical and quantitative studies. Unfortunately, advances are hampered by the significant time and effort that new researchers must invest to establish their research environments. In addition, existing researchers face challenges in reconstructing and modifying existing experiments, as well as in comparing new results against existing baselines. Members of the Center of Excellence for Software Traceability (CoEST) have been working to address these issues through identifying the Grand Challenges of Traceability, developing benchmarks, and constructing TraceLab, an extensible and scalable visual environment for designing and executing a broad range of traceability experiments.

Categories and Subject Descriptors

Traceability []

General Terms

Keywords

benchmarks, traceability,

1. INTRODUCTION

Requirements traceability is a critical component of any rigorous software development process. Among other things, it is used to demonstrate that a software design implements all of the specified software requirements, that all aspects of the design are traceable to software requirements, and that all code is linked to design specifications and test cases [2]. Unfortunately, despite its clear criticality, numerous case studies have shown that traceability can be difficult to accomplish in practice, primarily because creating and maintaining traceability links is time-consuming, costly, arduous, and error prone [11, 26]. These problems have created a compelling research agenda that includes qualitative studies of traceability users in industry [20, 9, 24] as well as a wide range of technical and quantitative studies addressing topics such as automated trace retrieval [4, 21, 17, 6, 7], link evolution [19], requirements satisfiability [14], and traceability across product lines [18]. Traceability research projects have been funded by government agencies including NSF, NASA, and EU commission, as well as private industries. As a result of these efforts, there have been several major advances in traceability techniques.

International Conference on Software Engineering 2011, Hawaii

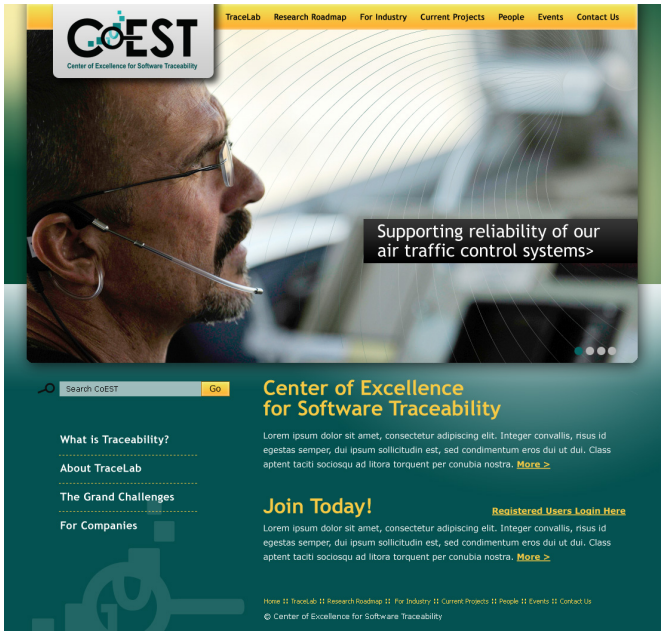

Figure 1: Center of Excellence for Software Traceability http://www.CoEST.org

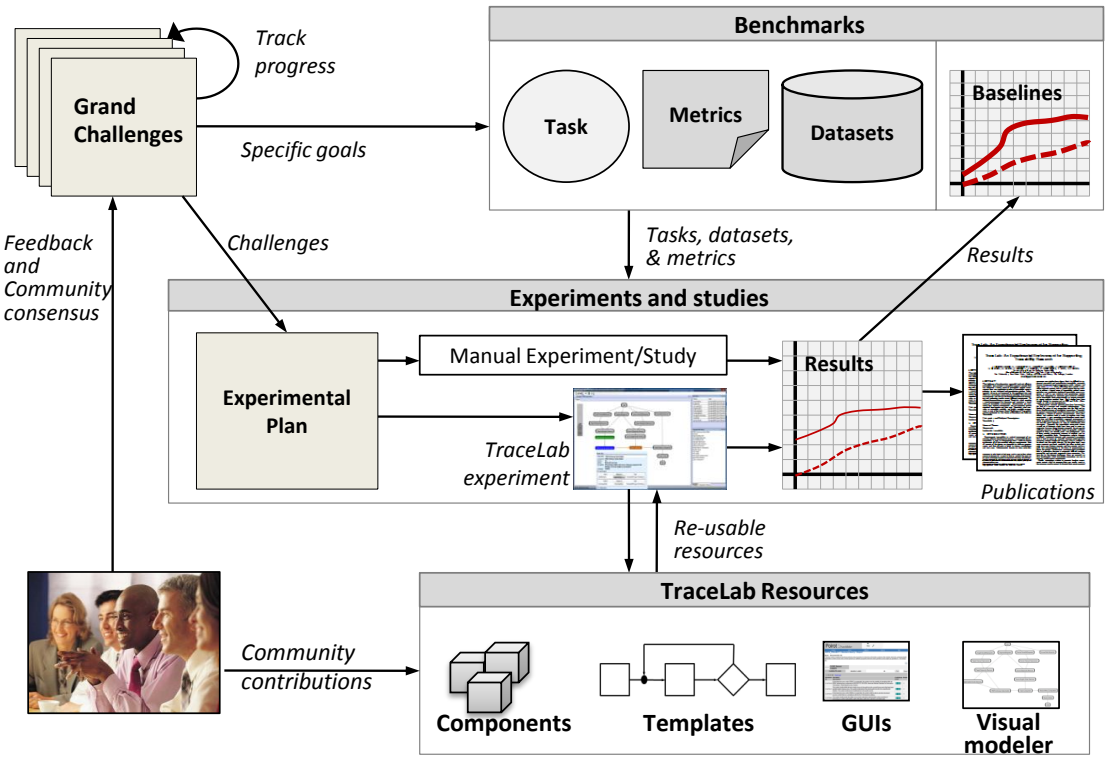

Figure 2: Elements of the Tracy Project

At the same time, advances have been hindered by the lack of a publicly available research infrastructure. New researchers entering the field must invest significant time to establish a research environment before they can become productive, while seasoned traceability researchers face challenges in comparatively evaluating their work against prior studies and must invest considerable time recreating prior experiments in order to test out new algorithms or new combinations of techniques.

2. CENTER OF EXCELLENCE

To address these challenges, the International Center of Excellence for Software Traceability (CoEST) was established in 2005 with the charge to "bring together traceability researchers and experts in the field, encourage research collaborations, assemble a body of knowledge for traceability, and develop new technology to meet tracing needs" [3] . CoEST membership currently includes academic, government, and industrial researchers from across the U.S. and Europe.

Since its inception, the CoEST has engaged in two primary projects, the **Grand Challenges of Traceability** (GCT) and the **Tracy** project. The grand challenges, which are discussed in greater detail in section 3 of this paper, provide a detailed road map of critical research and practice goals. The Tracy project is driven by the grand challenges, and as depicted in Figure 2, focuses on equipping the traceability research community through building research infrastructure, collecting and organizing datasets, establishing benchmarks, and developing a tool named TraceLab, which will provide support for designing and executing a broad range of traceability experiments.

3. GRAND CHALLENGES

Grand challenges, as their name suggests, are designed to challenge and inspire people to work towards achieving a difficult, yet significant goal. For example, in 1900, the mathematician David Hilbert formulated a list of important unsolved problems [13] which have engaged the creative thought of mathematicians ever since. More recently, in 1989, the US Federal High Performance Computing Program defined a modern day grand challenge as a "a fundamental problem in science or engineering, with broad applications, whose solution would be enabled by the application of high performance computing resources" [1]. Furthermore, a recent report entitled "Critical Code: Reproducibility for Defense" [8], highlighted traceability as one of the critical elements needed to build high-assurance systems. The report goes on to further detail the significant difficulties in implementing traceability effectively across large and complex software systems. To address such needs, CoEST members have therefore engaged in the task of identifying and defining the grand challenge of traceability. This project was initially launched in a series of workshop meetings funded by NASA and NSF, which produced the original "Grand Challenges" document [15]. Over the past three years, CoEST members have further organized the challenges so that they are more rigorously and systematically defined. The updated version (2.0) is now available [12] on the CoEST website.

The GCT version 2.0 identifies the single overarching traceability challenge as the need to achieve **ubiquitous trace-**

ability defined as "traceability which is always there, without having to think about getting it there." Furthermore, ubiquitous traceability is "neither consciously established nor sought; it is built-in and effortless. It has effectively disappeared without a trace" [12].

GCT 2.0 also identifies the following seven sub-challenges: (1) **Purposed**: Traceability is fit for purpose and supports stakeholder needs, (2) **Cost-Effective**: The return from using traceability is adequate in relation to the outlay of establishing it. (3) **Configurable**: Traceability is established as specified, moment-to-moment, and the rich semantics accommodate changing stakeholder needs. (4) **Trusted**: All stakeholders have full confidence in the traceability as it is created and maintained in the face of inconsistency, omissions, and change. (5) **Scalable**: An increasing number of artifacts are supported by traceability, of varying types and at varying levels of granularity, as traceability extends through the system life-cycle and across organizational and business boundaries. (6) **Portable**: Traceability information is exchanged, merged, and reused across projects, organizations, domains, product lines, and supporting tools. (7) **Valued**: Traceability is a strategic priority valued by all, where every stakeholder has a role to play and actively discharges his or her responsibilities.

As these challenges are specified at a very high level, each one of them is refined into a series of more concrete lower level goals, associated with specific research tasks. For example, the *configurable* sub-challenge has the associated research task "to develop tools, templates, and techniques of dynamic, heterogeneous and semantically rich traceability information models to guide the definition and provision of traceability"; while the *ubiquitous* challenge has the associated task of "total automation of trace creation and trace maintenance, with quality and performance levels superior to manual efforts."

It is anticipated that such research tasks will motivate researchers to engage in new projects and develop innovative solutions. A more complete discussion of the grand challenges of traceability, their associated goals and tasks, is found in the GCT version 2.0 [12] and at the CoEST website [3].

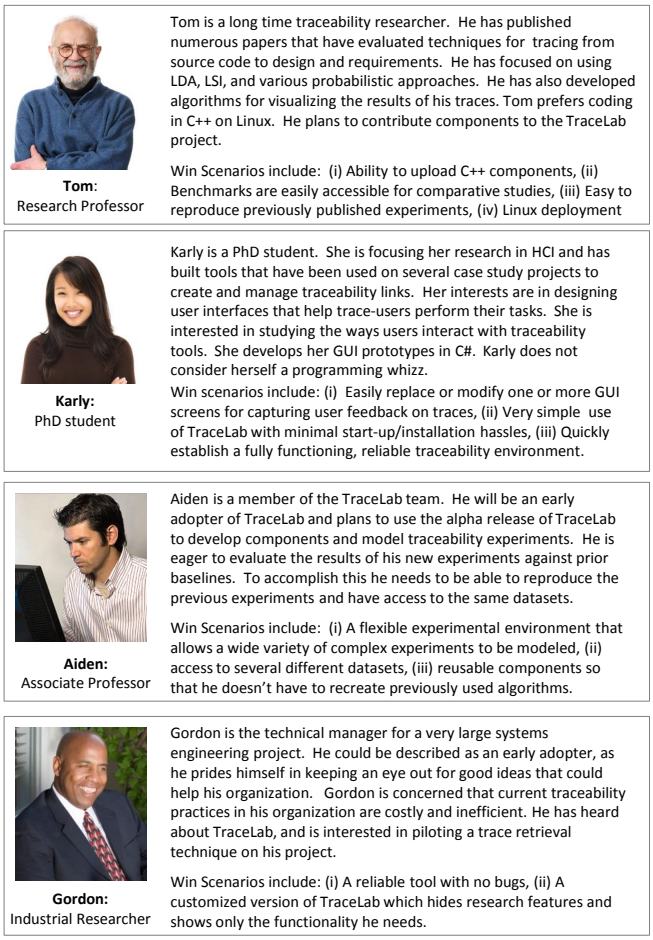

Figure 3: Personas respresenting four primary users of TraceLab

4. TRACKING PROGRESS

In addition to serving as a research guide, the GCT provide the opportunity for evaluating and tracking progress towards a specific goal. To that end, traceability researchers and practitioners are able to rate the importance, difficulty, and progress status of each research task. The CoEST website reports the average ratings for each research goal. Over time, we intend to show progression trends. Research tasks that are quantitative in nature can also be evaluated through the use of more formal benchmarks. These are discussed in section 4.1. Additionally, all publications are classified according to the research tasks they address, and are catalogued using a framework developed by Hayes and Dekhtyar [16] for comparing requirements tracing experiments.

4.1 Benchmarks

A benchmark is defined by the Merriam-Webster dictionary as a "standardized problem or test that serves as a basis for evaluation or comparison." Similarly, Oppenheimer, in his work on benchmarks for system dependability, claims that benchmarks provide researchers and system implementors with the means of quantifying design trade-offs and measuring and inspiring progress" [23]. Establishing benchmarks in the traceability community is therefore anticipated to facilitate a more rigorous comparison and analysis of experimental results.

We define a traceability benchmark to include (i) the description of a specific traceability task, for example "retrieve traces from requirements to code," (ii) the measures by which the effectiveness of the task will be evaluated, and (iii) the specific datasets on which the task is to be performed. Finally, over time as researchers conduct experiments against the benchmark and report their results, a set of baseline results will be established for use in future comparative experiments. The benchmarks are all traceable back to core goals in the grand challenges. This means that researchers can focus on the low-level tasks while retaining full accountability for how their work addresses higher level traceability goals [10].

Benchmarks can be created retrospectively or proactively. Retrospective benchmarks are established when it is observed that multiple traceability researchers have conducted related experiments, and there is a need to create a more rigorous and systematic approach for comparing and analyz-

ing their results. Benchmarks might be established proactively if the community as a whole, or even an individual researcher, recognizes an important traceability task, and proposes a benchmark in order to encourage research activity in that area. We present two candidate benchmarks, both of which are currently under consideration.

Benchmark #1:
Task: Automatically recover traces from documentation to code, with options to trace at the class or method level. No manual intervention is permitted.
Motivation: This is a traceability task which has attracted significant attention [4, 21, 22] by multiple research groups.
Relevant GCT challenge: RT1.3 (Inherent):Total automation of trace creation and trace maintenance, with quality and performance levels superior to manual efforts; RT6.9 (Trusted):Gain improvements in performance for the real-time automated recovery and capture or links.
Datasets: (i) LEDA (??), (ii) EasyClinic, (iii)Albergate.
Metrics: Standard trace retrieval metrics.
Status: Under review.

Benchmark #2:
Task: Automatically maintain links between requirements and UML class diagrams as the class diagrams evolve.
Motivation: Initial work performed in this area [19].
Relevant GCT challenge: RT1.3 (Inherent):Total automation of trace creation and trace maintenance, with quality and performance levels superior to manual efforts; RP5.7 (Trusted):Apply concepts from autonomic computing to explore self-healing traceability techniques and methods, covering diagnosis, repair actions and propagation, to apply at both the individual trace and collection of traces levels.
Datasets: (i) Pending, (ii) tbd, (iii) tbd.
Metrics: Under discussion
Status: Datasets missing

4.2 Comparative Frameworks

Benchmarks are most appropriate for comparing quantitative results; however, traceability researchers engage in a great variety of studies which are not so easily measurable. Basille, Shull, and Lanubile studied several different experimental frameworks for software engineering [5]. They pointed out that frameworks are useful for documenting key choices and rationales for experiments, and that, over the long-term, frameworks enable a shift in emphasis from questioning whether a specific approach is effective to determining what factors make it effective or ineffective. Huffman Hayes et. al. built on these ideas by proposing a framework for "developing, conducting, and analyzing experiments on requirements traceability" [16]. Their framework classifies results according to the way the experiment is defined (i.e. motivation, purpose, hypothesis etc), how the experiment is planned (i.e. dependent and independent variables, metrics, and data collection techniques), the way the experiment is realized or executed, and the way the results are interpreted. We further extend this framework to classify papers against specific research projects within the GCT. The next release of the CoEST website will include a self-reporting mechanism so that authors can classify their own papers against this framework when reporting results.

5. TRACELAB

Although defining the grand challenges, and providing benchmarks, evaluative frameworks, and datasets goes a long way towards establishing needed infrastructure for the traceability research community, it still falls short of helping new researchers to establish research environments or helping existing researchers to perform more rigorous evaluations and become more productive in their work. To this end, we are developing a visual experimental workbench, named TraceLab, for designing and executing traceability experiments. TraceLab is similar in some respects to existing tools such as Weka, MatLab, or RapidMiner, except that it is highly customized to support rigorous Software Engineering experiments as opposed to general data mining ones.

5.1 Features

As part of the TraceLab development process, we identified likely users and summarized their primary needs through the use of Personas [25]. Four of these personas and their anticipated usage scenarios are depicted in Figure 3. These personas represent (i) researchers who will use TraceLab to design and execute experiments, evaluate results against benchmarked baselines, exchange components, and to train students, (ii) PhD students, who will use TraceLlab to quickly establish their experimental environments and get acclimated to traceability research, (iii) developers, who will develop the initial releases of TraceLab or help to maintain TraceLab over the long-term, and finally (iv) industry adopters who may wish to pilot various traceability components on their own projects.

To meet these goals, TraceLab will include the following features: 1. A visual environment for designing and executing experiments. 2. An alternate scripting environment for supporting experiments. 3. A stand-alone player which compiles experiments for use in industrial pilots and other studies. 4. A component library which facilitates sharing of a wide variety of importers, pre-processors, algorithms, analyzers, etc. across the traceability community. 5. The ability for components to be written in a wide variety of languages including, C++, C#, and Java, and combined into a single experimental workflow. 6. A flexible workflow engine which support a wide variety of typical traceability experiments. 7. Interfaces to previously defined benchmarks, so that a researcher can design an experiment, run it against a benchmark, and compare results against existing baselines. 8. A scalable environment that supports experiments involving extremely large sized industrial datasets. 9. Portability across multiple operating systems including Windows, Linux, and Mac OS. 10. A simple installation process which allows new users to quickly download and install TraceLab. 11. An intuitive user interface which enables new users to execute basic experiments without any formal training.

5.2 Designing and Executing an Experiment

To use TraceLab a researcher can either retrieve an existing experiment or create a new one from scratch. Each experiment is represented as a precedence graph which determines the order in which components are executed. Components in the graph, exchange data through a data cache. At the start of an experiment, data is loaded into the data cache, named the *Borg*, using a special *importer* component. Although data may be stored in any user-defined data

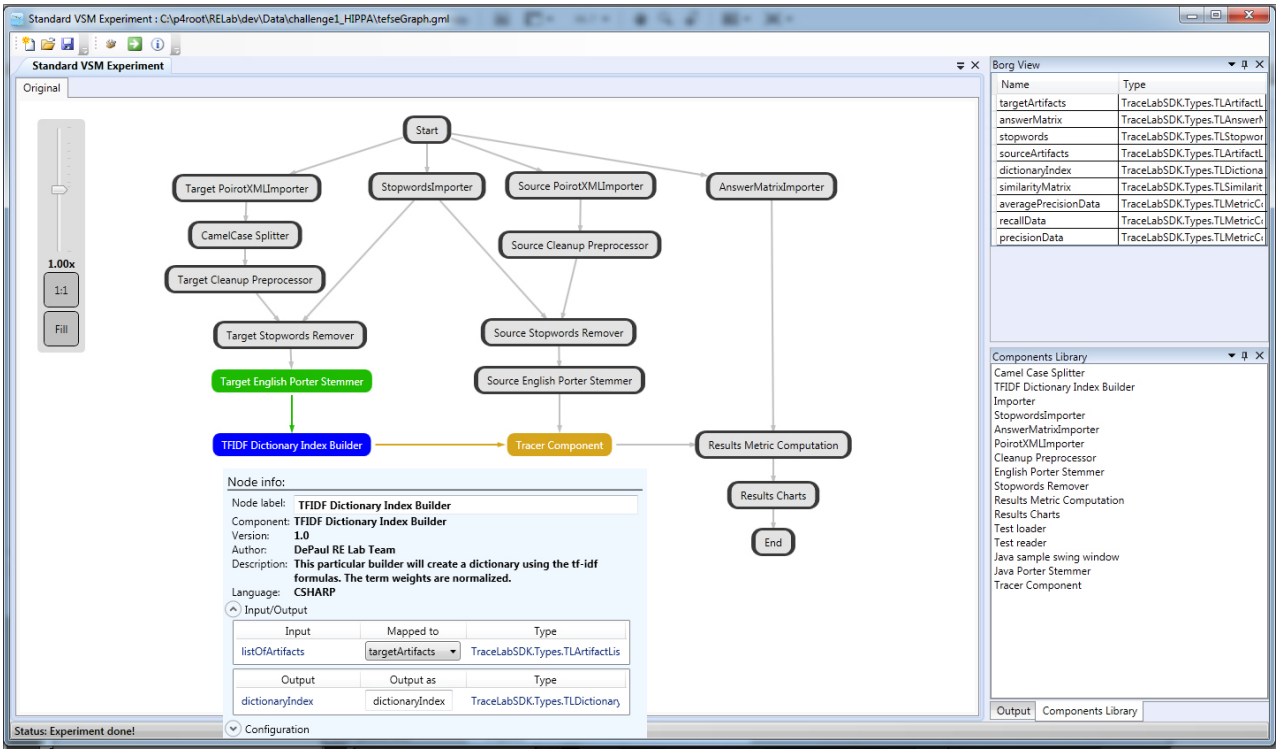

Figure 4: TraceLab Screen Shot

structure, TraceLab provides a fairly extensive set of predefined data types, which if used, can increase plug-n-play compatibility across components. A given component in the precedence graph, can use any of the data currently in the cache. In addition, any new data elements output by that component then become available for use by other downstream components.

Figure 4 depicts a basic experiment for tracing between requirements and java code using the Vector Space Model (VSM). This experiment uses four importers for importing java code methods (target), stop words, requirements (source), and the answer set against which results are evaluated. The imported artifacts are then preprocessed to split camelCase variable names, remove unwanted characters and stopwords, to stem terms to their root forms, and to produce a dictionary of terms.

A single component can be re-used in multiple places in the graph. In this example, the same *stop word remover* is used to process both the source and target artifacts, and is labeled accordingly as *Target Stopwords Remover* and *Source Stopwords Remover*. The workflow engine can execute components in any viable order allowed by the precedency relationships, including using parallel threads. In this example, a synchronization point occurs when the trace component is forced to wait for both the target artifacts and the source artifacts to complete processing before similarity scores are computed. Once traces are generated, their accuracy is evaluated against the answer set by the *results metric computation* component, and results are displayed in a GUI-based *Results chart*. Figure 4 shows a tool tip depicting information for the *Dictionary builder* component.

In order for a component to be integrated with TraceLab, the programmer needs to modify the code using TraceLab's API. First the component's metadata is defined, which includes a name, description, version, licensing, and configuration information. Secondly any data that the component needs to exchange with the Borg, is defined using the Input and Output functions from the IOSpec interface. Components in the experiment that are dependent on intermediate artifacts are required to use the Borg class interface to store and retrieve artifacts. Other data structures internal to a component do not require any special treatment. Once the experiment is designed, TraceLab can check the graph for problems such as illegal cycles or invalid import commands. Once validated, the experiment can be executed.

5.3 Experimenting against a benchmark

Although not implemented yet, TraceLab is designed to execute experiments against specified benchmarks. A benchmark controller is responsible for loading benchmark datasets, invoking the original experiment, and then collecting and reporting results. The researcher is responsible for creating data mappings between the benchmark controller and the original experimental design.

This is illustrated in Figure 5, which shows the same experiment from Figure 4, but this time run against benchmark # 1 to "Automatically recover traces from documentation to code, with options to trace at the class or method level." The experiment is automatically repeated for each of the datasets associated with the benchmark. For each dataset in turn, the data is loaded into the Borg. In this example, this means that the java methods are loaded as target artifacts and requirements as source artifacts. Stopwords are also loaded. This step replaces the traditional

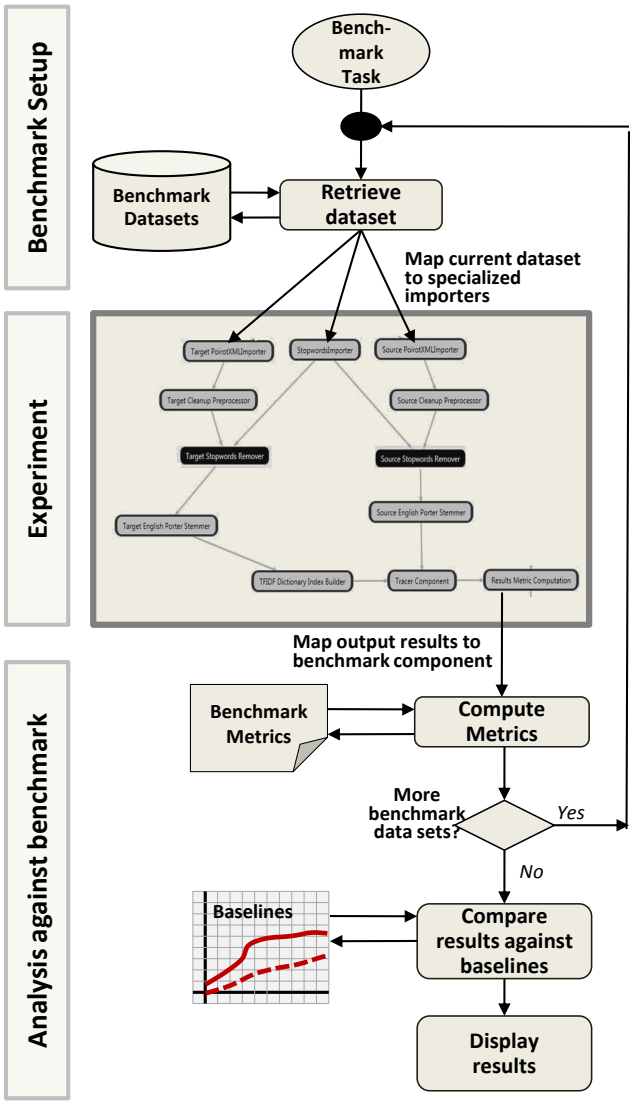

Figure 5: Benchmark experiment

import routine. Once data is loaded, the experiment is executed as per normal; however instead of displaying results and terminating, the results are passed back to the benchmark controller for analysis against the standard benchmark metrics. Once experiments have been run against all of the associated datasets, the results are integrated, analyzed, and reported against existing benchmark baselines.

5.4 Component Library

One of the primary goals of building TraceLab and developing the CoEST website, is to promote sharing across the traceability research community. Although TraceLab allows researchers to develop their own proprietary components,the intention is that many of these components will eventually be shared back into the community and available for future researchers to use. To this end, collaborators on the project are working to build sharable experiments and components which will be released as part of the TraceLab framework. Although TraceLab components are currently stored in an unstructured directory, future releases will provide a fully catalogued library of re-usable components. Furthermore, TraceLab will also ship with a set of predefined experimental templates representing standard experiments in the field. These templates will provide fully functioning experiments which can be customized by individual researchers through replacing or modifying components, or by restructuring the experimental workflow.

5.5 TraceLab Status

TraceLab is currently in the development phase with part of the functionality completed. To date the primary deliverables include the visual environment for designing and executing experiments, as well as the underlying data cache and facilities for importing components. The first alpha release is anticipated sometime around the Summer of 2011. TraceLab is developed in C# using Mono, which means that it is compilable to run on Windows, Mac, and Linux platforms. However, currently TraceLab's GUI has only been developed for Windows. The multi-platform release is anticipated by early 2012. The current release of TraceLab accepts components written in C#, managed memory versions of C++, and Java. Future releases will accept components written in C regardless of whether memory is managed or not.

TraceLab is designed as a desktop application, as this approach avoids any problems running experiments using datasets under non-disclosure agreements. However, in the future we will also provide a web-based version of TraceLab for use in pedagogical settings. Although in an incubation phase for the next two years, the TraceLab framework will be released as an open source product. Specific licensing decisions are still under consideration. At a recent collaborators meeting, researchers sketched out ideas for the experiments they would like to model and execute in TraceLab (see Figure 6), and demonstrated the potential flexibility of TraceLab to support a very broad range of experiments.

6. CONCLUSIONS

Moving a discipline forward towards more rigorous experimental standards and greater collaboration provides significant benefits to the community, but does not come without some amount of pain. One of the primary hindrances to standardization and benchmarking in the traceability community has been the difficulty of acquiring non-trivial datasets. While industry does sometimes make data available to individual research groups, it is almost always shared under a non-disclosure agreement, and can therefore never be used in a standard benchmark. Although for several research domains, open source projects may provide a wealth of useful data, this is not normally the case for traceability, as open source projects typically do not include any form of systematic requirements or traceability matrices. These kinds of artifacts are part and parcel of the large and complex systems for which traceability is most needed and must therefore be included in benchmark datasets. For research purposes, it is essential for datasets to come with validated traceability matrices. Although researchers can reconstruct such matrices on certain projects, it is difficult, if not impossible to do this in unfamiliar domains. It therefore remains a significant challenge for the traceability community to develop and/or acquire non-trivial datasets for use in benchmarking.

In conclusion, this paper has described the current state

Figure 6: Traceability Researchers and Developers exchange ideas for TraceLab experiments. MRI Meeting, DePaul University, February 2011

of CoEST's TRACY project, including the Grand Challenges, benchmarking, and TraceLab. As previously explained, CoEST was founded to serve the traceability research community and to foster advances in the field of traceability, As such, the work accomplished to date represents significant community effort and involvement. Further information concerning CoEST and opportunities for both academics and practitioners to get involved can be found at http://www.CoEST.org.

7. ACKNOWLEDGMENTS

The work described in this paper was primarily funded by the U.S. National Science Foundation grant #CNS-0959924.

8. REFERENCES

[1] High-Performance Computing Act of 1991, Jan 1991.

[2] U.S. Food and Drug administration, general principles of software validation. *U.S. Dept. of Health and Human Services*, 1(1), 2002.

[3] Center of Excellence for Software Traceability, http://www.traceabilitycenter.org, March 2008.

[4] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Trans. Software Eng.*, 28(10):970–983, 2002.

[5] V. R. Basili, F. Shull, and F. Lanubile. Building knowledge through families of experiments. *IEEE Trans. Software Eng.*, 25(4):456–473, 1999.

[6] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova. Best practices for automated traceability. *IEEE Computer*, 40(6):27–35, 2007.

[7] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker. A machine learning approach for tracing regulatory codes to product specific requirements. In *ICSE (1)*, pages 155–164, 2010.

[8] Committee for Advancing Software-Intensive Systems Producibility. *Critical Code: Software Producibility for Defense*. National Research Council, USA, 2011.

[9] D. Cuddeback, A. Dekhtyar, and J. Hayes. Automated requirements traceability: The study of human analysts. *Requirements Engineering, IEEE Intn'l Conference on*, 0:231–240, 2010.

[10] A. Davis. Requirements: A textbook example of goals displacement. *Requirements Engineering, IEEE Intn'l Conference on*, 0:xx, 2010.

[11] O. Gotel and A. Finkelstein. Contribution structures (requirements artifacts). In *RE*, pages 100–107, 1995.

[12] O. Gotel et al. The grand challenges of traceability 2.0. *Center of Excellence for Software Traceability*, 2(0), 2011.

[13] D. Hilbert. Mathematical problems. *Bulletin of the American Mathematical Society*, 1900.

[14] E. A. Holbrook, J. Huffman Hayes, and A. Dekhtyar. Toward automating requirements satisfaction assessment. In *RE*, pages 149–158, 2009.

[15] J. Huffman Hayes, J. Cleland-Huang, and A. Dekhtyar. The grand challenges of traceability. *Center of Excellence for Software Traceability*, 1(1), 2005.

[16] J. Huffman Hayes and A. Dekhtyar. A framework for comparing requirements tracing experiments. *Intn'l Journal of Software Engineering and Knowledge Engineering*, 15(5):751–782, 2005.

[17] J. Huffman Hayes, A. Dekhtyar, S. K. Sundaram, and S. Howard. Helping analysts trace requirements: An objective look. In *RE*, pages 249–259, 2004.

[18] W. Jirapanthong and A. Zisman. Xtraque: traceability for product line systems. *Software and System Modeling*, 8(1):117–144, 2009.

[19] P. Mäder, O. Gotel, and I. Philippow. Enabling automated traceability maintenance through the upkeep of traceability relations. In *ECMDA-FA*, pages 174–189, 2009.

[20] P. Mäder, O. Gotel, and I. Philippow. Motivation matters in the traceability trenches. In *RE*, pages 143–148, 2009.

[21] A. Marcus, J. I. Maletic, and A. Sergeyev. Recovery of traceability links between software documentation and source code. *Intn'l Journal of Software Engineering and Knowledge Engineering*, 15(5):811–836, 2005.

[22] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia. On the equivalence of information retrieval methods for automated traceability link recovery. In *Proc. of 18th IEEE Intn'l Conference on Program Comprehension (ICPC'10)*, pages 68–71, 2010.

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

[24] M. C. Panis. Successful deployment of requirements traceability in a commercial engineering organization...really. *Requirements Engineering, IEEE Intn'l Conference on*, 0:303–307, 2010.

[25] J. Pruitt and T. Adlin. *The Persona Lifecycle: Keeping People in Mind Throughout Product Design*. Morgan Kauffman, San Francisco, USA, 2006.

[26] B. Ramesh and M. Jarke. Toward reference models of requirements traceability. *IEEE Trans. Software Eng.*, 27(1):58–93, 2001.