A TraceLab-Based Solution for Creating, Conducting, and Sharing Feature Location Experiments

Bogdan Dit, Evan Moritz, Denys Poshyvanyk
Department of Computer Science
The College of William and Mary
Williamsburg, Virginia, USA
{bdit, eamoritz, denys}@cs.wm.edu

***Abstract*—Similarly to other fields in software engineering, the results of case studies involving feature location techniques (FLTs) are hard to reproduce, compare, and generalize, due to factors such as, incompatibility of different datasets, lack of publicly available implementation or implementation details, or the use of different metrics for evaluating FLTs. To address these issues, we propose a solution for creating, conducting, and sharing experiments in feature location based on TraceLab, a framework for conducting research. We argue that this solution would allow rapid advancements in feature location research because it will enable researchers to create new FLTs in the form of TraceLab templates or components, and compare them with existing ones using the same datasets and the same metrics. In addition, it will also allow sharing these FLTs and experiments within the research community. Our proposed solution provides (i) templates and components for creating new FLTs and instantiating existing ones, (ii) datasets that can be used as inputs for these FLTs, and (iii) metrics for comparing these FLTs. The proposed solution can be easily extended with new FLTs (in the form of easily configurable templates and components), datasets, and metrics.**

***Index Terms*–TraceLab, feature location, experiments, benchmarks**

I. INTRODUCTION

There are numerous case studies in software engineering that present contradictory results for a given technique applied (reproduced) on different projects by other researchers [1]. As a consequence, the same technique, which was shown to work in one experimental setting, but not in others, may cast some doubts on the external validity of such empirical studies and applicability of that technique [1, 2].

Feature location is empirical in nature, with case studies as the predominant research method for evaluating the results [3]. However, there are several factors that impact the reproducibility, comparison, and generalizability of many of the existing empirical results. Some of these factors, which serve as our motivation, are enumerated next.

First, the datasets used in the evaluation are different in many cases. In a feature location survey by Dit et al. [4] it was shown that only three out of 60 papers (5%) used the same datasets for evaluating their proposed FLTs. In all the other cases, the techniques were evaluated using datasets from different software systems, different versions, or even different subsets of data points from those subject systems.

Second, the actual implementation of feature location techniques (FLTs) is not always made publicly available, and this is the case with almost all the feature location papers summarized in our previous work [4]. The unavailability of the implementation of a FLT makes it difficult to compare new FLTs with existing ones. For example, assuming that a researcher is proposing a new FLT, she will have to compare her technique against (relevant) existing ones. However, implementing previous FLTs is not only time consuming, but also error prone, as some of the implementation details or settings may not be explicitly stated in the research paper that introduced the existing techniques. Out of the 60 surveyed feature location papers, only 23 (38%) compared their techniques against a very small subset of existing ones [4].

Third, the FLTs are evaluated using different metrics. For example, some techniques are evaluated using precision, recall or the effectiveness measure [4]. In some cases, even if the techniques use the same metric, the results may be at different levels of granularity (e.g., class, method, or statement level) and thus, not directly comparable [4].

All these previously enumerated factors (i.e., different datasets, lack of implementation details for existing techniques and different metrics used in the evaluation) make FLTs hard to compare and reproduce in empirical studies. In addition, the number of feature location research papers increased considerably in the last years. According to the recent survey [4], the total number of feature location research papers that use textual, dynamic or static information, or a combination of these types of information was just four in the year 2000, 15 in the year 2005 and 54 in the year 2010. This means that between 2000 and 2005 there were only 11 papers published, whereas between 2005 and 2010 there were 39 feature location papers published. Note that these numbers are for research papers only and do not include tool papers or posters.

Given the large number of new FLTs in the last years, and the fact that the area of feature location research becomes more mature, the standards for publishing new techniques are also increasing. For example, new techniques are oftentimes expected to be compared against existing state-of-the-art ones using sound statistical methods. These requirements are necessary to ensure the progress in this and many other research areas, but at the same time they result in substantial effort for developing and publishing new techniques. This is

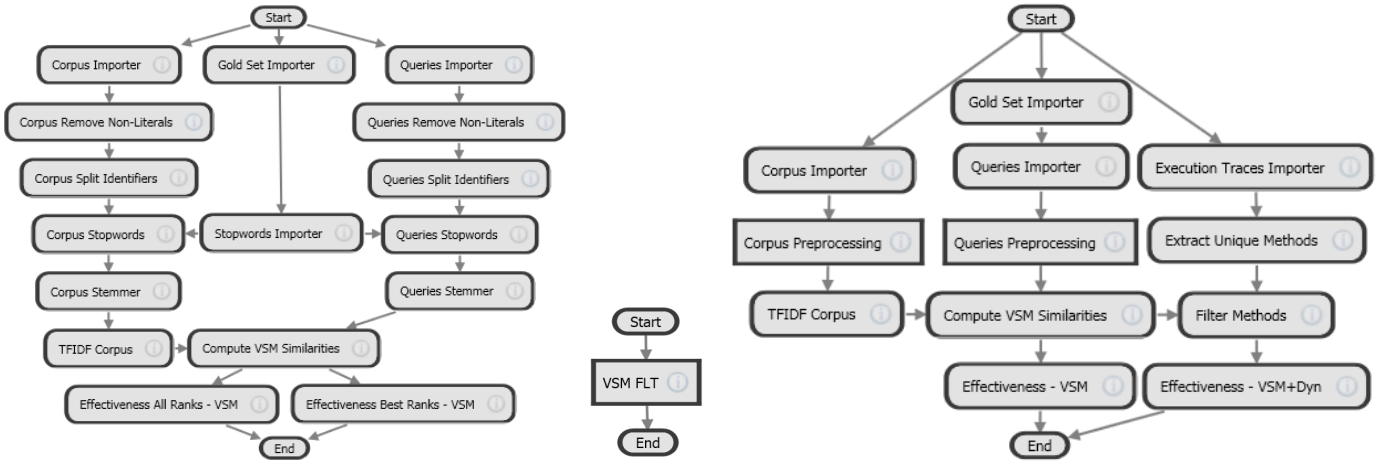

Figure 1 (a) The VSM feature location experiment in TraceLab using primitive components (rectangles with rounded corners); (b) The VSM feature location experiment in TraceLab as a composite component (rectangle with sharp corners); (c) One possible implementation of the VSMDyn FL experiment in TraceLab that was adapted from the VSM FL experiment from Figure 1 (a)

because a large portion of research effort is dedicated to reimplementation and comparison with other techniques, which is non-trivial in many cases [4].

In order to address these problems, we are proposing a solution to uniformly create, conduct, and share feature location experiments using the TraceLab framework [5, 6, 7]. This solution contains templates and components for creating feature location experiments, datasets for running these experiments and metrics for evaluating FLTs. More details and the data are available on our online appendix[1].

II. TRACELAB BASICS

In this section we describe the TraceLab framework [5, 6, 7] and discuss some of its important characteristics that make it suitable for feature location research.

A. TraceLab

TraceLab [5, 6, 7] is a framework for creating, running and sharing experiments using a visual modeling environment. TraceLab is funded by the National Science Foundation and is developed at DePaul University with collaborating partners at Kent State University, University of Kentucky, and the College of William and Mary. Currently TraceLab is still in beta version, but it is expected to be released for public use by June 2012, with many stability and performance improvements, as well as a large set of reusable components. TraceLab is designed to support research in the area of traceability link recovery. However, due to its design qualities and characteristics it is well suited to be adapted to other software engineering tasks [6], such as feature location. Some of these characteristics are discussed in the next subsection.

B. TraceLab Characteristics

TraceLab's visual modeling plug-and-play environment was designed to support a wide range of experiments in traceability link recovery research. A TraceLab experiment (see Figure 1 (a)) is a graphical representation of a directed graph, in which nodes are TraceLab components and directed edges are dependencies between components.

1) Components. A primitive component (see rectangular shapes with rounded corners in Figure 1 (a)) is a computational unit that takes as input data from external sources (e.g., files) or data produced by other components and produces data that will be used by other components, or data that can be saved to external sources (e.g., files). Components can be written in C#, Java, or other memory managed programming languages. The developer of a component must specify metadata information, such as the name, description, author, etc. of the component, as well as the data types required as input and the data types produced as output.

A composite component is composed of a set of primitive components, and has the same functionality as the primitive components it encapsulates. An example of a composite component is presented in Figure 1 (b). The "VSM FLT" composite component encapsulates all the functionality of the experiment presented in Figure 1 (a).

A special type of component is a decision component (see component "More datasets?" in Figure 2 (b)), which allows the possibility of changing the control flow of the experiment, based on some values. In other words, a decision component is similar to an *if* statement in any procedural programming language, but it also allows to create loop structures by manipulating the control flow. For example, the decision component "More datasets?" from Figure 2 (b), along with the primitive components "Get next dataset" and "datasetIndex++" are used to iterate through multiple datasets in the experiment.

2) Datatypes. The datatypes for the input and output can be chosen from either predefined TraceLab datatypes or they can be user defined datatypes. The datatypes already implemented in TraceLab consist of primitives (e.g., lists, strings, integers), and complex datatypes specific to some areas of software engineering, such as similarity matrices, co-

[1] http://www.cs.wm.edu/semeru/data/TraceLab-feature-location/

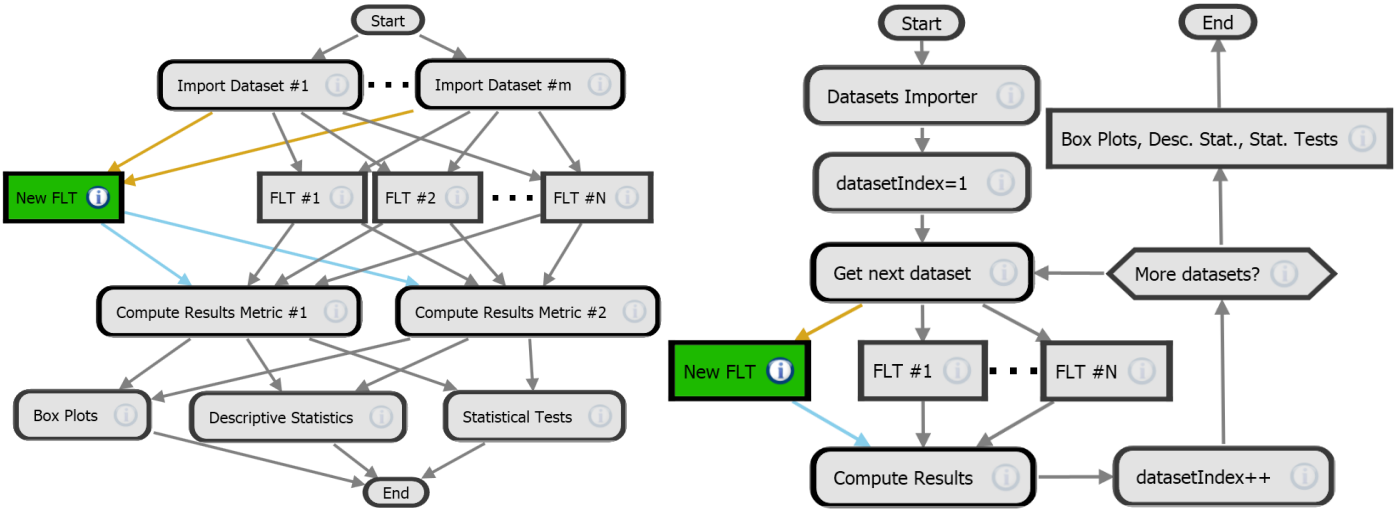

Figure 2 (a) Example of comparing the proposed FLT (the composite node in green color) with existing FLTs on the same datasets and metrics, using boxplots, descriptive statistics and statistical tests; (b) An alternative example for comparing the proposed FLT (the composite node in green color) with existing FLTs on the same datasets and metrics, using boxplots, descriptive statistics and statistical tests. This example is suited when there are multiple datasets to be used as input. The multiple datasets are loaded at the beginning and a decision node (i.e., the "More Datasets?" node) allows iterating through all the datasets before giving the control flow to the component that displays and compares the results

occurrence matrices, etc. The TraceLab framework allows users to develop their own datatypes to fit their needs. In addition, users can develop custom datatype viewers for their datatypes, which allows them to view the data stored in the variables of that datatype while the experiment is running, or after the experiment finished. All the variables of the predefined TraceLab datatypes or the user developed datatypes can be viewed at any time in the TraceLab Workspace. In addition, the TraceLab Workspace allows components to exchange information with each other, by storing data to or loading data from the Workspace.

3) Dependencies. The dependencies (see arrows in Figure 1 (a)) ensure that components do not execute until the components depended upon are executed. This precedence order of components ensures that the data required by a component was already computed by the components it depends on. For example, in Figure 1 (a), the component "Compute VSM Similarities" that computes the VSM similarities between a corpus of methods and a set of external queries cannot execute until the components "TFIDF Corpus" and "Queries Stemmer" are executed, which are responsible for generating the required data from the corpus and queries respectively. However, if components are independent from each other, they can be executed in parallel. For example, "Corpus Importer" and "Queries Importer" can be executed in parallel because they do not depend on each other.

III. Feature Location in TraceLab

In this section we describe the process of creating, conducting, and sharing feature location experiments in TraceLab. In addition, we provide some details about the artifacts that we released for transitioning feature location research to TraceLab.

A. Creating Experiments

In some cases new components need to be developed because the FLT being introduced may be so unique or novel that there are no other similar components that can be reused. These are *user defined components* and can be easily implemented in C# or Java, or any other programming languages that support memory management.

In the other cases, using TraceLab can substantially reduce the effort for creating new FLTs by reusing *templates*, *primitive components* or *composite components* [5].

1) Templates. Templates are TraceLab experiments that can be easily adapted to create new FLTs. For example, the vector space model (VSM) FLT presented in Figure 1 (a), can be easily adapted to implement another FLT (called VSMDyn) that is similar to SITIR [8], which combines information retrieval results with dynamic information from execution traces. SITIR uses only a subset of methods (i.e., the methods from the execution trace) for computing the similarities between the developer query and the methods, whereas an information retrieval technique such as VSM computes the similarities between a query and all the methods in the corpus. In other words, it is easy to implement VSMDyn FLT by keeping the existing components of the VSM experiment and adding some components that handle the dynamic information from execution traces. Figure 1 (c) shows one possible implementation for the VSMDyn FLT that was adapted from the VSM FLT (see Figure 1 (a)). The new components that were added to the VSMDyn FLT (see Figure 1 (c)) to handle the dynamic information are "Execution Traces Importer", "Extract Unique Methods" and "Filter Methods", which imports the execution traces into the Workspace, preprocesses them to extract the unique methods from each execution trace, and eliminates from the final results the methods that do not appear in the execution trace.

Note that in Figure 1 (c) the composite component "Corpus Preprocessing" has the same functionality as the primitive components "Corpus Remove Non-Literals", "Corpus Split Identifiers", "Stopwords Importer", "Corpus Stopwords" and "Corpus Stemmer" from Figure 1 (a). Analogously is for the composite component "Queries Preprocessing" from Figure 1 (c). In addition, the composite components "Corpus Preprocessing" and "Queries Preprocessing" are the same component that was instantiated twice in the experiment, to use different data. Furthermore, the primitive components "Effectiveness - VSM" and "Effectiveness - VMS+Dyn" from Figure 1 (c) are the same component that was instantiated twice.

Using multiple instances of a component in the same experiment shows the flexibility and adaptability of TraceLab to easily create experiments, and it also encourages researchers to create components that are general enough to be reused by others.

2) Primitive Components. When developing a new FLT that is not very similar to others, it may be the case that there are no suitable templates to use, but there may be reusable components that can be utilized. For example, the new technique might use primitive components for importing data (e.g., the "Corpus Importer" or "Gold (answer) Set Importer" components from Figure 1 (a)) or for computing similarities (e.g., "Compute VSM Similarities" component in Figure 1 (a)) between two sets of artifacts, such as methods, bug reports, documentation, etc.

3) Composite Components. Another example of reusable components is a composite component, which is a set of components that are grouped under a single component. For example the components "Box Plots", "Descriptive Statistics" and "Statistical Tests", from Figure 2 (a) could be grouped under one single composite component with the same functionality, called "Box Plots, Descriptive Statistics, Statistical Tests" (see Figure 2 (b)). Composite components are easier to handle, especially while designing complex experiments, and can be considered as procedures that encapsulate statements (primitive components) in procedural programming languages.

B. Conducting and Comparing Experiments

Once a new FLT is created by adapting templates or using existing components or user defined components, it can be run from inside TraceLab, and its results could be displayed. However, the results of the new FLT would not be very indicative unless they are compared against the results of existing FLTs using the same datasets and the same evaluation metrics. Figure 2 (a) shows a TraceLab experiment that would allow comparing multiple FLTs on the same datasets, using the same metrics. The template should be customized to meet the requirements of the user and the new FLT, but the general steps in creating a new experiment are as follows.

First, from the existing datasets a subset (or all) should be chosen as input for the evaluation. For instance, datasets #1 through #m were chosen in Figure 2 (a). In case there are numerous datasets, the user does not have to instantiate an "Import dataset" component for each dataset. Instead, she has the choice to use a different experiment (see Figure 2 (b)) that has a component that loads multiple datasets (see Figure 2 (b), "Datasets Importer") based on a configuration file defined by the user. Once the datasets have been loaded, the experiment iterates over the datasets and uses them to compute the FLTs results. The iteration over the datasets is possible using the decision node "More datasets?" (see Figure 2 (b)) which decides which nodes should get the control flow next, based on some variables values. The logic of the decision node is specified by the user when designing the experiment. In this example, the logic is to give the control flow to the component "Get next dataset" if there are still datasets that still need to be used as inputs for the FLTs, or give the control flow to the composite component "Box Plots, Descriptive Statistics, Statistical Tests" to compare the results, after all the datasets have been used as inputs.

Second, from the already implemented FLTs, a subset (or all) should be chosen as the baseline for the comparison of the results. Note that in some cases, the choice of FLTs may be restricted by the datasets chosen. For example, if a chosen dataset does not contain execution traces, or any other dynamic information, then no FLTs that use dynamic information can be chosen as baseline for comparison. In Figure 2 (a), the chosen FLTs are FLT_1 through FLT_n.

Third, given the choice of datasets and FLTs, the appropriate metrics for comparison are chosen. In other words, the datasets, the FLTs, and the metrics should be selected appropriately, so that they are consistent with one another. For example, if a dataset has an incomplete set of methods related to a feature (i.e., incomplete gold set), then the recall metric cannot be used, but the precision or effectiveness [4] measure could be used instead. For instance, $metric_1$ and $metric_2$ were chosen to evaluate the results in Figure 2 (a).

Fourth, the methods for comparing and analyzing the results of the FLTs need to be selected. The results can be compared (i) visually, using box plots or precision-recall curves, (ii) using descriptive statistics (e.g., minimum, 25th percentile, median, 75th percentile, maximum, average, standard deviation, etc.), or (iii) using statistical tests, such as the Wilcoxon signed-ranked test, in order to determine if one FLT produces results that are statistically significant. In Figure 2 (a), the components for comparing the techniques are located on the next to last line and are "Box plots", "Descriptive statistics" and "Statistical tests".

The benefits of using TraceLab to conduct feature location experiments are clearly depicted in Figure 2 (a), as the effort to develop a new FLT and compare it against existing ones is reduced considerably. In addition, the development of the new FLT can be significantly increased using existing components.

C. Sharing Experiments

Similarly to the way primitive components in TraceLab are grouped into a single composite component, a FLT can be encapsulated into a single composite component that has the same functionality as the group of primitive or composite components that created it (see the "VSM FLT" composite component from Figure 1 (b)). This has the advantage of not only allowing for an easier manipulation of FLTs when designing experiments in TraceLab's visual modeling environment, but also sharing such composite components with the research community.

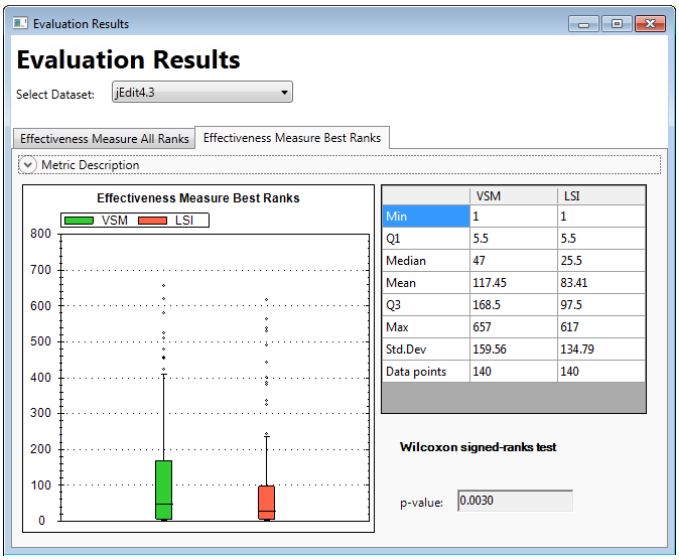

Figure 3 The results of the evaluation of the effectiveness measure between the VSM and LSI FLTs: box plots (left), descriptive statistics (middle right) and statistical test (lower right). The datasets can be chosen from the combo box at the top, and the different metrics can be chosen using the tabs. Note that 10 of the data points with highest scores (i.e., outliers) for the jEdit dataset were eliminated from the results to properly display the box plots

TraceLab already provides the functionality for creating composite components for easy sharing, using a wizard-based interface. The user only needs to specify the metadata information (e.g., the composite component's name, author, description, etc.), as well as the input and output data types and the configuration parameters, by clicking the appropriate check boxes in the wizard. The exported data consists of a TraceLab description file in XML format, which can be used in the same way as other components.

A TraceLab feature for exporting experiments is currently being developed and is estimated to be ready by June 2012. This feature would allow to export an entire experiment, including primitive and composite components, their implementation (e.g., assemblies or byte code), and datasets. The exported experiment will have support for referencing other exported experiments or datasets. For example, if the research community publishes datasets, and FLTs in the form of exported experiment, a new researcher who uses those datasets and the existing FLTs to evaluate her new technique would simply reference them in her experiment, instead of including them. This would be possible, as each experiment or dataset has a Globally Unique Identifier (GUID), which allows the referencing to be robust to version changes. In other words, if an experiment was tested on a specific implementation (or version) of the components, and on specific versions of the datasets, the experiment will produce the same results, regardless if there are new versions of those components or datasets, because the experiment will reference the versions of the components and datasets it originally used by their GUID.

D. Artifacts for Feature Location in TraceLab

In this section we describe some of the software artifacts (datasets, templates, components, and metrics) that we made available in order to transition feature location research to TraceLab. For more detailed explanations and updates please refer to our online appendix.

1) Datasets. We make available five datasets for the purpose of testing new experiments involving new FLTs. These datasets are for the following five Java software systems: ArgoUML [2], Eclipse [3], JabRef [4], jEdit [5] and muCommander [6]. These datasets were used in previously published case studied on FL [9, 10], as well as in case studies on impact analysis [11].

The datasets contain issues (e.g., bugs, features, patches, etc.) extracted from their issue tracking system, and these issues have associated with them gold sets, queries and execution traces.

The gold sets are set of methods related to the functionality described in the textual description of the maintenance task (i.e., issue). The gold sets were extracted from the patches submitted to the issue tracking system (for Eclipse), or were generated by mining SVN repositories (for the other four systems) and matching SVN commit log messages to issue IDs.

The queries are textual descriptions of the maintenance tasks (i.e., issues) and are composed of the title of the issue as well as its description.

The execution traces were collected at method level granularity by reproducing the steps described in the issue description on an instrumented version of the system.

A more detailed description of these datasets and their format is presented in the survey [4] and its online appendix[7].

2) Templates and Components. We also provide the source code and executable code of the components that were used in building four FLTs. In addition, we also provide their templates.

The first two templates are based on information retrieval, namely the Vector Space Model FLT (see Figure 1 (a)) and Latent Semantic Indexing [12] (LSI) FLT. The LSI FLT uses a component that imports the similarities between queries and methods, which were computed externally. A feature of TraceLab is being developed that would allow access to powerful frameworks and libraries such as GenSim[8] or R[9], which would allow TraceLab experiments to benefit from the complex functionality implemented in these libraries.

The other two templates combine information retrieval and dynamic analysis and are similar to the SITIR approach [8]. These FLTs are VSMDyn (see Figure 1 (c)) and LSIDyn.

Our online appendix provides more resources on how to get started with developing new components and using existing ones.

3) Metrics. In addition to the datasets and the FL templates and components, we provide metrics in the form of TraceLab components that can be used to evaluate FLTs.

---

[2] http://argouml.tigris.org/
[3] http://www.eclipse.org/
[4] http://jabref.sourceforge.net/
[5] http://www.jedit.org/
[6] http://www.mucommander.com/
[7] http://www.cs.wm.edu/semeru/data/benchmarks/
[8] http://radimrehurek.com/gensim/
[9] http://www.r-project.org/

These metrics are the effectiveness of all ranks and the effectiveness of best ranks. The effectiveness of all ranks returns the rank of all the methods from the gold set associated with a feature, whereas the effectiveness of best ranks returns the first position (i.e., the best) among all the methods from the gold set for each feature.

Moreover, we created components that would allow for a fair comparison between different FLTs evaluated on the same datasets (see Figure 3). These components can compare the effectiveness of FLTs based on box plots (see Figure 3, left), descriptive statistics (see Figure 3, middle right) and statistical tests, such as the Wilcoxon signed-ranked test (see Figure 3, lower right). The developer can also choose to see the results from different datasets (see combo box in upper left corner of Figure 3), or the results of different metrics (see tabs in upper left corner of Figure 3).

The components related to metrics and comparisons between FLTs are currently in beta version, but we are working closely with TraceLab's development team to ensure that those components are available by June 2012.

IV. CONCLUSIONS

In this paper we presented a TraceLab-based solution for facilitating rapid advancements in feature location research. More specifically, we provided the details for an environment that allows researchers to create, conduct, compare, and share feature location techniques in the form of TraceLab experiments. We made available an initial set of templates and components that can be easily adapted or used to create new FLTs, as well as datasets and metrics that can be used to reproduce previous work and/or evaluate new FLTs. We envision TraceLab as a framework that can be used to conduct feature location research, and facilitate the progress in the feature location area. Hence, we will continually expand the repository from our online appendix with new datasets and components. In addition, we see a great potential for TraceLab to be used in support of other software engineering areas, such as impact analysis, detection of duplicate bug reports, recommending expert developers, etc.

ACKNOWLEDGMENT

This work is supported in part by the United States NSF CNS-0959924 grant. Any opinions, findings and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors. We would also like to acknowledge the team of researchers from DePaul University: Jane Cleland-Huang, Ed Keenan, Adam Czauderna, Greg Leach, and Yonghee Shin. This work would not have been possible without their continuous support on the TraceLab project.

REFERENCES

[1] T. Menzies and M. Shepperd, "Special Issue on Repeatable Results in Software Engineering Prediction", *Empirical Software Engineering (ESE)*, vol. 17, no. 1-2, 2012, pp. 1-17.

[2] A. Arcuri and L. C. Briand, "A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering", *in Proceedings of 33rd IEEE/ACM International Conference on Software Engineering (ICSE'11)*, Honolulu, Hawaii, USA, May 21-28 2011, pp. 1-10.

[3] T. J. Biggerstaff, B. G. Mitbander, and D. E. Webster, "The Concept Assignment Problem in Program Understanding", *in Proceedings of 15th IEEE/ACM International Conference on Software Engineering (ICSE'94)* May 17-21 1994, pp. 482-498.

[4] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature Location in Source Code: A Taxonomy and Survey", *Journal of Software Maintenance and Evolution: Research and Practice (JSME)*, vol. doi: 10.1002/smr.567, 2012, pp. to appear.

[5] J. Cleland-Huang, A. Czauderna, A. Dekhtyar, G. O., J. Huffman Hayes, E. Keenan, G. Leach, J. Maletic, D. Poshyvanyk, Y. Shin, A. Zisman, G. Antoniol, B. Berenbach, A. Egyed, and P. Maeder, "Grand Challenges, Benchmarks, and TraceLab: Developing Infrastructure for the Software Traceability Research Community", *in Proceedings of 6th ICSE2011 International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE2011)*, Honolulu, HI, USA, May 23 2011.

[6] J. Cleland-Huang, Y. Shin, E. Keenan, A. Czauderna, G. Leach, E. Moritz, M. Gethers, D. Poshyvanyk, J. H. Hayes, and W. Li, "Toward Actionable, Broadly Accessible Contests in Software Engineering", *in Proceedings of 34th IEEE/ACM International Conference on Software Engineering (ICSE'12), New Ideas and Emerging Results Track*, Zurich, Switzerland, June 2-9 2012, pp. to appear 4 pages.

[7] E. Keenan, A. Czauderna, G. Leach, J. Cleland-Huang, Y. Shin, E. Moritz, M. Gethers, D. Poshyvanyk, J. Maletic, J. H. Hayes, A. Dekhtyar, D. Manukian, S. Hussein, and D. Hearn, "TraceLab: An Experimental Workbench for Equipping Researchers to Innovate, Synthesize, and Comparatively Evaluate Traceability Solutions", *in Proceedings of 34th IEEE/ACM International Conference on Software Engineering (ICSE'12)*, Zurich, Switzerland, June 2-9 2012, pp. to appear 4 pages.

[8] D. Liu, A. Marcus, D. Poshyvanyk, and V. Rajlich, "Feature Location via Information Retrieval based Filtering of a Single Scenario Execution Trace", *in Proceedings of 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07)*, Atlanta, Georgia, November 5-9 2007, pp. 234-243.

[9] B. Dit, M. Revelle, and D. Poshyvanyk, "Integrating Information Retrieval, Execution and Link Analysis Algorithms to Improve Feature Location in Software", *Empirical Software Engineering*, vol. doi: 10.1007/s10664-011-9194-4, 2012, pp. to appear.

[10] M. Revelle, B. Dit, and D. Poshyvanyk, "Using Data Fusion and Web Mining to Support Feature Location in Software", *in Proceedings of 18th IEEE International Conference on Program Comprehension (ICPC'10)*, Braga, Portugal, June 30 - July 2 2010, pp. 14-23.

[11] M. Gethers, B. Dit, H. Kagdi, and D. Poshyvanyk, "Integrated Impact Analysis for Managing Software Changes", *in Proceedings of 34th IEEE/ACM International Conference on Software Engineering (ICSE'12)*, Zurich, Switzerland, June 2-9 2012, pp. to appear 10 pages.

[12] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by Latent Semantic Analysis", *Journal of the American Society for Information Science*, vol. 41, no. 6, 1990, pp. 391-407.