University of Kentucky TraceLab Component Similarity Matrix Voting Merge

Jared Payne
Computer Science
University of Kentucky
Lexington, Kentucky, USA
jared.payne@uky.edu

Jane Huffman Hayes
Computer Science
University of Kentucky
Lexington, Kentucky, USA
hayes@cs.uky.edu

***Abstract*—We present one of the University of Kentucky TraceLab components, Similarity Matrix Voting Merge. We highlight some particularly interesting aspects of the component such as challenges faced when developing it. We discuss the challenges encountered when setting up unit testing for the component. We provide an example of the component being used in a TraceLab experiment. We provide a link for download of the component.**

Keywords— TraceLab, components, traceability, unit testing

I. Introduction

TraceLab [1] is an experimental framework developed under the National Science Foundation's Major Research Instrumentation program. The framework facilitates the building, sharing, and composition of components into larger experiments. Originally aimed at improving replicability and decreasing start up time for traceability research, TraceLab has since been broadened to include components and support of other activities such as software testing and can be used for empirical research in any subfield of software engineering.

The University of Kentucky has developed a number of TraceLab components. These are available at http://selab.netlab.uky.edu/homepage/pages/TraceLabCatalogue/TracelabCatalogue/componentspage.html. This paper addresses one specific component which is provided in the catalogue, Similarity Matrix Voting Merge.

The paper is organized as follows. Section II describes the component and the research paper on which it is based. Section III discusses what the component does and how it works. Section IV discusses the approach used to introduce unit testing to this and other components developed by the University of Kentucky and challenges encountered during the process. Finally, Section V discusses the component as part of a larger TraceLab experiment.

II. Technique Integration for Requirements Assessment

Dekhtyar et al. [2] examined the notion of combining tracing techniques into a "committee" in order to evaluate a given trace matrix. Specifically, they applied five different techniques (Vector Space Model using term frequency-inverse document frequency weighting [3] (VSM-tf-idf), Latent Semantic Indexing [4] (LSI), Latent Dirochlet Analysis [5] (LDA), Chi-squared based Keyword Extraction [6] (KE), Probabilistic Information Retrieval [7] (IR)) to the CM-1 dataset, using a variety of schemes for the committee of techniques to vote on a link to be vetted (majority, supermajority, consensus). They found that the techniques were very good at keeping true links (high recall) when using majority and supermajority schemes. When these voting schemes were filtered, they were able to "beat" VSM tf-idf [2]. Regarding precision, the consensus rule only failed to discard 40% of the false positives (whereas other schemes kept 90%+ of the false positives) [2]. All three of the voting rules had better precision than VSM tf-idf except for the majority rule which only outperformed VSM tf-idf when unfiltered [2]. This experiment inspired our component. It should be noted that Poshyvanyk et al. [8] also used the idea of combining tracing techniques for feature identification.

The Similarity Matrix Voting Merge component allows a collection of similarity matrices, each generated by different tracing techniques, to be combined using different voting or decision rules. The aim is to generate better trace matrices using the committee of techniques. The component is agnostic as to how the similarity matrices are generated. The component applies a user-provided voting percent threshold to generate the merged matrix. For example, if four matrices are being combined and the user-selected threshold is 50%, any links appearing in two or more of the matrices will be written to the final matrix. Next, we discuss the development of the component.

III. Development of the Component

The component was developed using C# in the Visual Studio IDE. It was added to a solution of pre-existing TraceLab components. This solution was created in order to streamline the testing and installation of the components, as well as to place code shared among them in a common library. The solution contains 15 projects, nine of which are TraceLab components. The other six comprise the common library and unit test projects. There are also an additional 12 Java components stored alongside these components, but they are not a part of the C# solution.

The component functions by reading in a directory of files containing instances of the TLSimilarityMatrix class that have been converted to a plaintext format. This format is read in as a

We thank NSF for partially funding this work under grants CCF-1511117 and CICI 1642134.

collection of lines, with each line containing three space-separated elements: a source identifier, target identifier, and similarity score. The component deserializes these files back into instances of the TLSimilarityMatrix class. Then, the component merges the matrices into a single matrix while removing links that appear in a percentage of the matrices less than the given threshold. Link scores in the output matrix are the average score given across all the input matrices. Finally, the matrix is stored in the TraceLab workspace.

The component can be configured with an input directory and a score threshold in TraceLab before runtime. Its configuration options as displayed in TraceLab are shown in Fig. 1. A screenshot of its code inside the component solution is shown in Fig. 2 at the end of the paper. Next, we address testing.

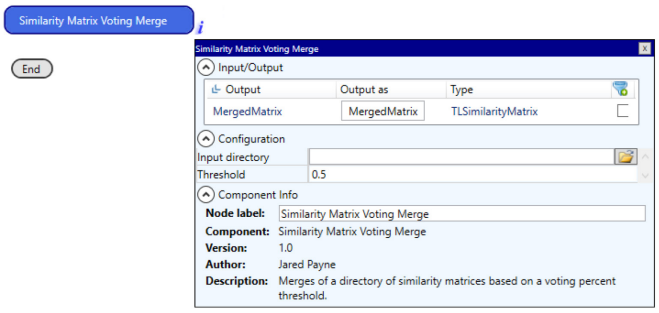

Fig. 1. The configuration options for the Similarity Matrix Voting Merge component.

IV. Testing of the Component

This section discusses the current state of testing of the Similarity Matrix Voting Merge component and our other TraceLab components, followed by the challenges encountered in introducing unit tests to the component solution.

A. Current State of the Component Catalogue Tests

As of this writing, five of the nine components in the component solution are coupled with their own unit test project, named in the format ComponentProjectName.Tests by convention. These projects contain an average of eight unit tests per tested component, with a total of 40 unit tests for the solution. All tests are passing. These tests typically check to ensure that components are correctly performing their intended transformation upon the data being examined in TraceLab. They also check that the proper exceptions are thrown and that correct, helpful information is provided to the user whenever a component is not properly configured.

As an example, the test cases for the Similarity Matrix Voting Merge component have been provided in Table I. Currently, no unit tests have been implemented for any of the Java components in the catalogue. Possible steps moving forward would be to either use JUnit to test each of these components individually, or to convert these components to C# so that they can be added to the component solution and tested simultaneously with the other components.

TABLE I. Test Cases for the Component

Class	Test Name
SimilarityMatrix VotingMerge Component	Compute merges similarity matrices
	Component configuration is custom type
SimilarityMatrix VotingMerge Configuration	Threshold default value is fifty percent
	Throws argument exception when directory path does not exist
	Argument exception provides directory error message when directory path does not exist
	Throws argument exception when threshold is greater than one
	Throws argument exception when threshold is less than zero
	Argument exception provides threshold error message when threshold is out of range
	Throws argument null exception when directory path is null
	Argument null exception provides directory error message when directory is null

B. Challenges

One of the challenges in developing this TraceLab component and maintaining our catalogue of existing components was ensuring that the components function properly and without bugs. Because TraceLab components are intended to be used in a variety of tracing experiments and behave similarly to code blocks in a visual programming language, ensuring that they are stable and execute predictably is of the utmost importance.

To accomplish this, Visual Studio's built-in unit testing framework for .NET framework projects was utilized. Doing so allowed development to move towards a more test-driven approach, as the unit tests for all the components could be quickly executed at any time. Since the unit tests are also provided alongside the components, this also allows other users of the components to verify the integrity of the code before they are installed.

The second issue encountered when testing TraceLab components was that many of them tend to manipulate files in the filesystem. Many of our own components do so as well, which posed an additional problem for testing because it introduced the unit tests to the volatile state of a filesystem, thereby often rendering the tests non-repeatable.

To resolve this issue, a package named System.IO.Abstractions was retrieved from the NuGet package manager and integrated into the solution. This package abstracts the .NET framework's concept of a filesystem from that of a static class to an interface. This way, multiple implementations of a filesystem can be created such that a class can instead be given a specific instance type of a filesystem to use. In our case, we used two types: a type representing the actual filesystem; and a virtual, mock filesystem that can be initialized during testing.

For our TraceLab components to have access to this mock filesystem, an internal constructor was added to each component class that accepted an additional filesystem parameter. This way, components could be constructed with a mock filesystem during unit testing, and TraceLab could continue to use its expected constructor while running experiments. When the original constructor is used, the component is initialized to use the real filesystem. When the mock filesystem is being used, it is reset

We thank NSF for partially funding this work under grants CCF-1511117 and CICI 1642134.

for each test case, and files with data for testing the component are then created. Doing so ensured that the unit tests return the same result every time.

V. Experimental Context for the Component

The TraceLab experiment file that we built for using the component utilizes several pre-existing components to import data, to perform pre-processing, and to calculate results. In order to use our voting merge component in an experiment, a multitude of components that use tracing techniques to analyze text must be used in order to generate the similarity matrices. Because the voting merge component uses a directory for input, any arbitrary number of matrices can by merged. Fig. 3 shows a portion of an example experiment where two similarity matrices are being merged by the component.

The component is provided for download at http://selab.netlab.uky.edu/homepage/pages/TraceLabCatalogue/TracelabCatalogue/TraceLabComponents.zip. We request that this paper be cited when using our work in any future published work.

Acknowledgment

We thank NSF for partially funding this work under grants CCF-1511117 and CICI 1642134.

References

[1] Ed Keenan, Adam Czauderna, Greg Leach, Jane Cleland-Huang, Yonghee Shin, Evan Moritz, Malcom Gethers, Denys Poshyvanyk, Jonathan Maletic, Jane Huffman Hayes, Alex Dekhtyar, Daria Manukian, Shervin Hossein, and Derek Hearn, "TraceLab: An experimental workbench for equipping researchers to innovate, synthesize, and comparatively evaluate traceability solutions," in 2012 34th International Conference on Software Engineering (ICSE), 2012, pp. 1375–1378.

[2] Alex Dekhtyar, Jane Huffman Hayes, Senthil Karthikeyan Sundaram, Elizabeth Ashlee Holbrook, Olga Dekhtyar: Technique Integration for Requirements Assessment. RE 2007: 141-150

[3] Baeza-Yates, Ricardo A., Berthier A. Ribeiro-Neto: Modern Information Retrieval. ACM Press / Addison-Wesley, 1999.

[4] Deerwester, S., S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman, Indexing by Latent Semantic Analysis, J. Am. Soc. Information Science, vol. 41, no. 6, pp. 391-407, 1990.

[5] Heinrich, Gregor, "LDA-J Library" Code available at http://www.arbylon.net/projects/.

[6] Huffman Hayes, J., Dekhtyar, A., Holbrook, E.A., Sundaram, S., Dekhtyar, O., Will Johnny/Joanie Make a Good Software Engineer?: Are Course Grades Showing the Whole Picture?, in Proc., Conference on Software Engineering Education and Training (CSEET), 2006, pp. 175 - 182.

[7] Cleland-Huang,J., C.K. Chang, G. Sethi, K. Javvaji, H. Hu, and J Xia. Automating speculative queries through event-based requirements traceability. Proc. International Requirements Engineering Conference (RE'02), 2002.

[8] Poshyvanyk, D., Gueheneuc Y., Marcus, A., Antoniol, G., and Rajlich, V. Combining Probabilistic Ranking and Latent Semantic Indexing for Feature Identification in Proc..ICPC 2006, pp. 137-148.

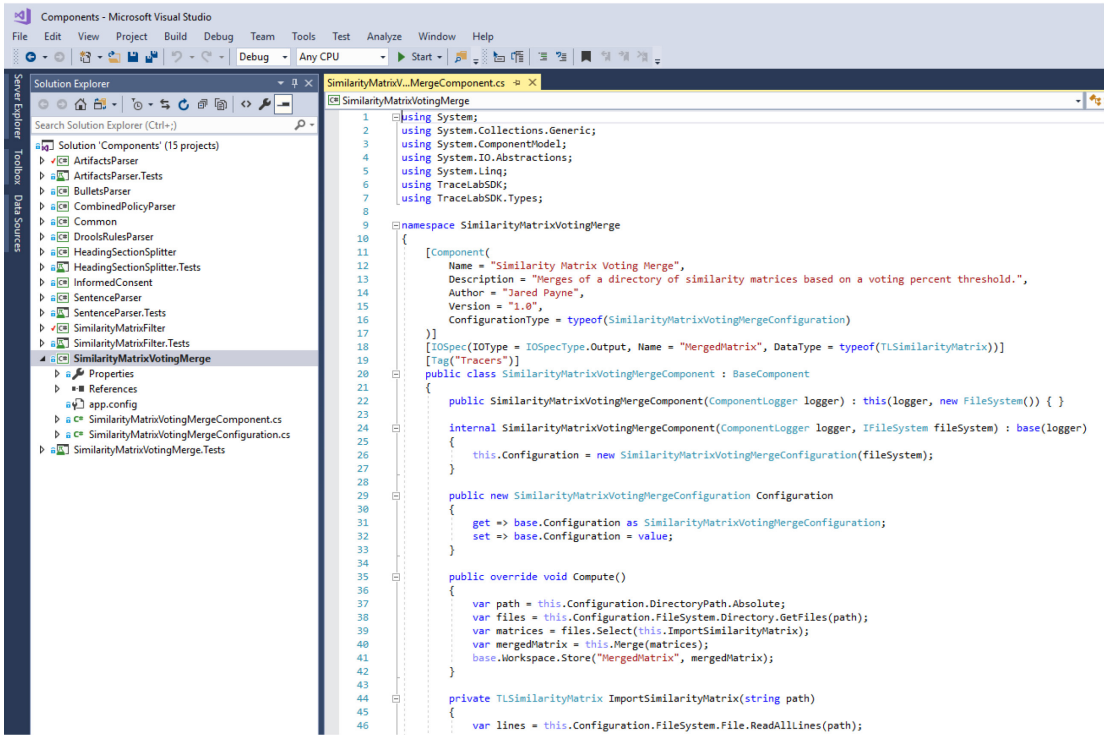

Fig. 2. The TraceLab components solution in Visual Studio 2017. The code for the Similarity Matrix Voting Merge component class is displayed.

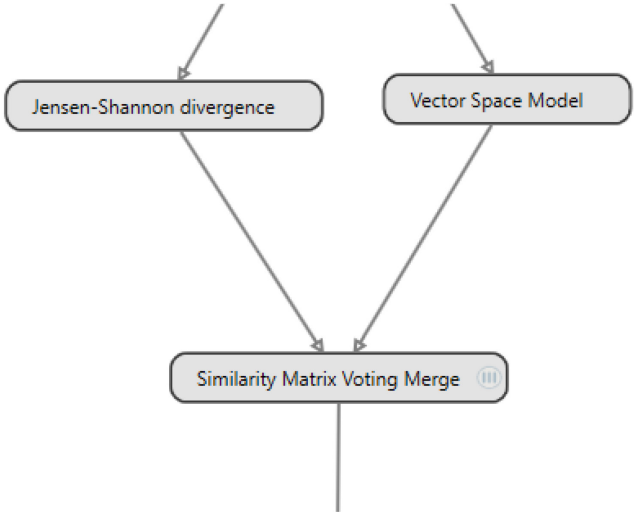

Fig. 3. The Similarity Matrix Voting Merge component being used in a TraceLab experiment. Two similarity matrices are being merged.