FLAT^3: Feature Location and Textual Tracing Tool

Trevor Savage
College of William and Mary
Department of Computer Science
P.O. Box 8795
Williamsburg, VA 23187-8795
1+757-221-3455
tcsava@cs.wm.edu

Meghan Revelle
College of William and Mary
Department of Computer Science
P.O. Box 8795
Williamsburg, VA 23187-8795
1+757-221-3467
meghan@cs.wm.edu

Denys Poshyvanyk
College of William and Mary
Department of Computer Science
P.O. Box 8795
Williamsburg, VA 23187-8795
1+757-221-3476
denys@cs.wm.edu

ABSTRACT
Feature location is the process of finding the source code that implements a functional requirement of a software system. It plays an important role in software maintenance activities, but when it is performed manually, it can be challenging and time-consuming, especially for large, long-lived systems. This paper describes a tool called FLAT^3 that integrates textual and dynamic feature location techniques along with feature annotation capabilities and a useful visualization technique, providing a complete suite of tools that allows developers to quickly and easily locate the code that implements a feature and then save these annotations for future use.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – enhancement, restructuring, reverse engineering, and reengineering

General Terms
Documentation, Design

Keywords
Program comprehension, concept location, information retrieval, dynamic analysis, software evolution and maintenance

1. INTRODUCTION
During software maintenance, it is very common for developers to search for source code that is relevant to their task. When their task pertains to modifying, extending, or adding functionality, their search is known as *feature* (or *concept*) *location* [1, 2]. For example, assume a developer working on an open source text editor needs to modify the *file saving* feature. The developer first needs to find the existing source code that implements *file saving* before he can make any changes. If the developer has never worked with this particular feature before, he will not know where to begin and may spend a great deal of time and effort manually searching for the feature's source code before being able to make any changes.

To aid developers in this situation, automated feature location techniques have been proposed to reduce the amount of time and effort spent searching for a feature's implementation. Some of these approaches employ information retrieval (IR) to search a body of text, such as source code, for sections that are relevant [9]. Other techniques analyze dynamically-collected execution traces to identify a feature's implementation [6, 20]. IR and dynamic analysis have also been combined to form hybrid feature location techniques [1, 8].

To make these feature location approaches more accessible to developers, we have created FLAT^3, the **F**eature **L**ocation and **T**extual **T**racing **T**ool. It is an Eclipse[1] plug-in that supports three well-established feature location techniques: 1) information retrieval (IR), 2) dynamic collection of execution traces, and 3) a combination of IR and dynamic tracing known as SITIR (**Si**ngle **T**race + **I**nformation **R**etrieval) [8]. Feature location via IR involves textually searching a project's source for code that is similar to a query that describes a feature. Dynamic feature location entails running the software and invoking the feature of interest to capture a trace of the source code that was executed. FLAT^3 also implements SITIR, which integrates textual and dynamic feature location techniques so that they can be used together effectively.

In addition to providing support for multiple feature location techniques, FLAT^3 also supports annotating and saving relevant search results. The tool permits developers to create and name features to which the source code implementing them can be linked. This feature mapping functionality allows developers to save their feature location results and avoids the need to repeatedly search for a given feature's implementation.

FLAT^3 makes two significant contributions that current feature location tools do not provide. Existing tools generally support one way of searching (i.e., IR only or dynamic tracing only). FLAT^3 makes both the IR and dynamic techniques available, and it also integrates them. FLAT^3's second contribution is its feature annotation function. While there are some tools that provide this functionality [14, 16], they are not coupled with feature location techniques, and existing feature location tools do not provide

ICSE '10, May 2-8, 2010, Cape Town, South Africa

[1] http://www.eclipse.org/

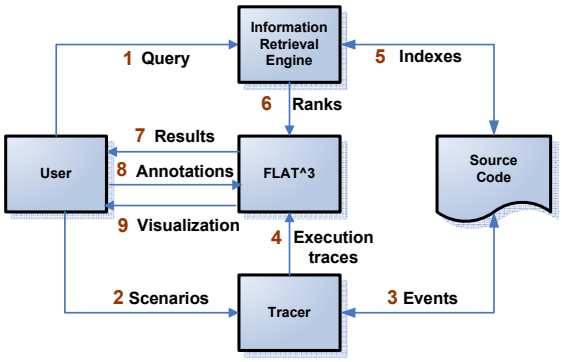

Figure 1. Overview of the architecture of FLAT^3.

mechanisms for saving the mappings of features to source code. FLAT^3 is a complete suite of feature location, annotation, and visualization tools.

2. FLAT^3

FLAT^3 is implemented as an Eclipse plug-in. Figure 1 gives an overview of FLAT^3's architecture. The tool combines the functionality of several existing libraries and applications. It uses information retrieval from the Lucene[2] library to locate and rank code by similarity to a user's query. FLAT^3 also uses MUTT[3] to capture execution traces of feature-specific scenarios and test cases. FLAT^3's feature annotation capability is based on ConcernMapper[4] and ConcernTagger[5], Eclipse plug-ins that allow for the creation of concern (feature) models and for source code to be linked to features. By integrating these existing tools, FLAT^3 provides developers with a way to easily search for features' implementations and annotate their findings for future reuse. Based on the annotations, FLAT^3 can also visualize the location of a feature's source code across a system's classes using a map metaphor similar to the one used in AspectBrowser[6]. FLAT^3's features are described in detail below.

2.1 Textual Feature Location

The first way in which FLAT^3 allows developers to perform feature location is textually. FLAT^3 textually searches for a feature's source code by leveraging the Lucene information retrieval library. To use this functionality, developers open the FLAT^3 Features view in Eclipse and click on the search toolbar button. This action opens a dialog box into which developers can enter a query that describes the feature they are trying to find, such as "file saving." After the query is issued, Lucene indexes Eclipse's workspace if it has not already been indexed. Indexing involves creating a document for each method and field consisting of all the words used in the method or field. Keywords and common stop words (e.g., "the" and "a") are removed. Also, words are split (e.g., "compoundIdentifier" becomes "compound" and "identifier") and stemmed (e.g., "searching" becomes "search"). Each document is converted to a vector, as is the query. Then, all the document vectors are compared to the query vector to determine their similarity, and a score is assigned to each method or field based on that similarity.

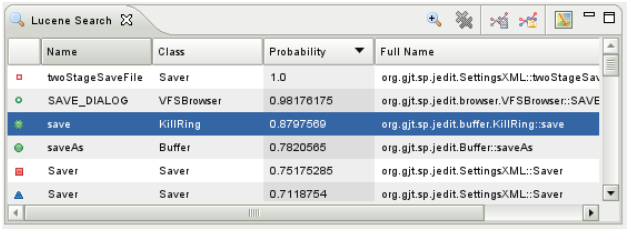

Figure 2. FLAT^3's Search/Trace Results view with a list of classes, methods, and fields returned by Lucene sorted by similarity to a query.

Figure 2 shows FLAT^3's Search/Trace Results view, listing the results returned by Lucene for the "file saving" query from the source code of jEdit[7], an open source text editor. The results include the method or field's name, class, a score of how similar it is to the query, it's fully qualified name, and any features with which it has been previously annotated (not visible in the figure). The results are ordered by their relevance to the query. Developers can double click on a result to view that method or field's source code. If a result is deemed to be relevant to the feature of interest, it can be annotated in this view, as will be explained in Section 2.4. Developers can also refine their results by searching within the original results with a new query.

2.2 Dynamic Feature Location

In addition to textual feature location, developers can also use FLAT^3 to locate features dynamically. This approach to feature location uses MUTT, a tracing tool based on the Java platform debugger architecture[8] (JPDA). MUTT runs a subject program on its own Java virtual machine and collects a trace of runtime method calls. What is unique about MUTT is the user can control when to turn tracing on and off with a button.

To perform dynamic feature location in FLAT^3, developers first determine a scenario or test case that invokes the desired feature. For instance for the *file saving* feature, a scenario would be to start jEdit, open a file, make changes, save the file, and exit. To collect an execution trace, developers right click on the class that contains the project's main method and select "Trace with MUTT," as in the first part of Figure 3. This will launch the program along with a separate window with a start/stop button to control tracing, as in the second part of Figure 3. The start button should be clicked just before the feature is invoked, and tracing should be stopped just after the feature's behavior completes. All methods that were executed between the start and stop interval are collected in a trace. Once developers are done tracing, they can close the application and return to FLAT^3 to find a listing of the methods executed by the scenario. The listing is very similar to Lucene's results (see Figure 2) with the exception that no similarity scores are given. Developers can browse these results to find relevant methods instead of searching the full source code of the system. Just as with Lucene's results, double clicking a method from the trace opens its source code for viewing. Traces can be saved and loaded again instead of having to be recollected.

[2] http://lucene.apache.org/java/docs/index.html
[3] http://sourceforge.net/projects/muttracer/
[4] http://www.cs.mcgill.ca/~martin/cm/
[5] http://www1.cs.columbia.edu/~eaddy/concerntagger/
[6] http://cseweb.ucsd.edu/~wgg/Software/AB/
[7] http://www.jedit.org/
[8] http://java.sun.com/javase/technologies/core/toolsapis/jpda/

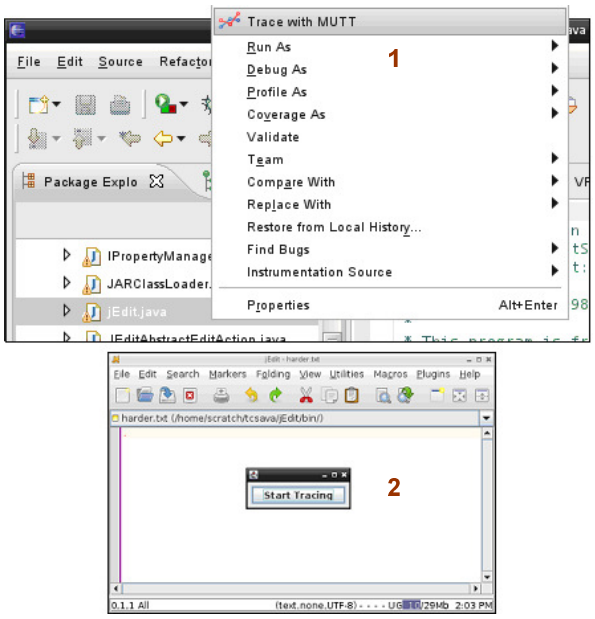

Figure 3. Invoking MUTT on jEdit in Eclipse (1) and jEdit running with MUTT's tracing control button (2).

2.3 Integrated Feature Location

FLAT^3 allows for the integration of its two separate feature location techniques. Since dynamic feature location in FLAT^3 is likely to return many methods, to narrow the results, it can be integrated with textual feature location following the SITIR approach [8]. After collecting an execution trace for a feature, IR is used to rank only the invoked methods instead of all of the methods in the system. In FLAT^3, after collecting a trace with MUTT, Lucene can be used to textually search only within the executed methods by clicking the "Refine Search" button. This opens a dialog in which developers can enter a query, causing Lucene to compute similarity scores for the methods in the trace as described in Section 2.1. The methods are indexed beforehand, and only similarities are computed at this point. After the scores are calculated, developers are presented with a list of the trace's methods ranked by their similarity to the query.

Combining two types of feature location techniques employs more sources of information to find a feature's implementation than a standalone approach. Dynamic tracing acts as a filter to IR by limiting the methods that are ranked to only those that are executed. This idea was first introduced in the PROMESIR approach [10] and further refined in SITIR [8].

2.4 Annotating Features

Once a feature's source code has been found using either textual feature location, dynamic feature location, or their combination, it can be annotated and saved with FLAT^3. In the Features view, features can be created and given a name. Then classes, methods, and fields can be associated with a feature from any of the results views by right clicking on the method and selecting "Link" and the name of the feature to which the code belongs. Code can also be mapped to features through Eclipse's package explorer, outline view, and editor. Code can be mapped to multiple features.

Figure 4 shows the Features view, listing the code associated with the *Line Number* feature. A feature's methods are grouped hierarchically by class. Code can be removed from methods by right clicking on them and selecting "Unlink" and the name of the feature. Features and their mappings are saved and can be revisited when FLAT^3 is reopened. Saving the mappings of source code to features acts as a form of documentation, making it easier to keep track of and modify features and their implementations [15].

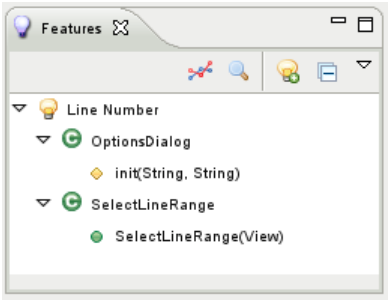

Figure 4. FLAT^3's Features view showing code associated with the *Line Number* feature.

2.5 Visualization

FLAT^3 also provides a visualization functionality that shows the distribution of a feature or search results across files. The visualization is accessible by right clicking on a feature and selecting "Visualize feature..." or by clicking the "Visualize" button after obtaining results from Lucene or MUTT. FLAT^3 uses the same map metaphor as AspectBrowser [18] to visualize the location of aspects in files. Figure 5 shows an example of the FLAT^3 visualization. Each box represents a class, and each row of pixels in a class' box corresponds to a section of code. If the row is highlighted in red, it means that code is associated with the feature or present in the search results. If Lucene's results are visualized, the shade of the row of pixels indicates the degree of similarity of that section of code to the user's query. This visualization gives developers a global idea of where a feature of interest is implemented.

3. RELATED WORK

FLAT^3 is based on several existing tools. The Lucene library provides full-text searches, MUTT collects execution traces, and ConcernTagger and ConcernMapper [16] lend the ability to annotate and save feature mappings. These functionalities are integrated in FLAT^3. There are other existing tools that implement either feature location or annotations, but not both. IRiSS [12], JIRiSS [11], and Google Eclipse Search [13] are tools that support feature location via Latent Semantic Indexing (LSI)

Figure 5. FLAT3's visualization view showing classes from a Lucene search that has code similar to the query. The color of highlighted rows indicates the degree of similarity of the code to the feature.

[4], an advanced IR method. FLAT^3 relies on Lucene, so it is faster than LSI-based tools. While none of these tools allow for the saving of located feature code, FEAT [14] and ConcernTagger do. However, these tools rely on manual feature location. There are several other feature location tools such as STRADA [5] which uses dynamic information; JRipples [3] and Suade [19] which use static analysis; Find-Concept [17] which uses natural language processing; and Dora [7] which uses textual and static analysis. However, FLAT^3 is unique in that it combines textual and dynamic feature location with annotations and visualization.

4. CONCLUSION

FLAT^3 is a novel tool suite for feature location. It is implemented as an Eclipse plug-in and combines the functionality of a number of existing tools in one easy-to-use application. FLAT^3 allows developers to perform feature location textually and dynamically, to save their results for future reference, and to visualize the dispersion of features or search results throughout a project. Future work on FLAT^3 includes making it more robust to be able to index large source code bases, trace larger programs, and save and update annotations for evolving programs. A user study to evaluate the tool's usability is also planned.

5. ACKNOWLEDGEMENTS

FLAT^3 incorporates source code from the open source tools ConcernMapper, ConcernTagger, and MUTT. It also inherits visualization ideas from AspectBrowser. Alison Smith and Scott Underwood contributed to an earlier version of the FLAT^3 tool. This work is supported by NSF CCF-0916260 and United States AFOSR grant number FA9550-07-1-0030. Any opinions, findings and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

6. AVAILABILITY

FLAT^3 is free and publicly available for academic use. The most recent version of the plug-in, its source code, and a user manual are available at http://www.cs.wm.edu/semeru/flat3.

7. REFERENCES

[1] Antoniol, G. and Guéhéneuc, Y. G., "Feature Identification: An Epidemiological Metaphor", IEEE Trans. on Software Engineering, vol. 32, no. 9, Sept. 2006, pp. 627-641.

[2] Biggerstaff, T. J., Mitbander, B. G., and Webster, D. E., "The Concept Assignment Problem in Program Understanding", in Proc. of ICSE'94, May 17-21 1994, pp. 482-498.

[3] Buckner, J., Buchta, J., Petrenko, M., and Rajlich, V., "JRipples: A Tool for Program Comprehension during Incremental Change", in Proc. of IEEE International Workshop on Program Comprehension, 2005, pp. 149-152.

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

[5] Egyed, A., Binder, G., and Grunbacher, P., "STRADA: A Tool for Scenario-Based Feature-to-Code Trace Detection and Analysis", in Proc. of ICSE'97, pp. 41-42.

[6] Eisenbarth, T., Koschke, R., and Simon, D., "Locating Features in Source Code", IEEE Trans. on Software Engineering, vol. 29, no. 3, March 2003, pp. 210 - 224.

[7] Hill, E., Pollock, L., and Vijay-Shanker, K., "Exploring the Neighborhood with Dora to Expedite Software Maintenance", in Proc. of International Conference on Automated Software Engineering, Nov. 2007, pp. 14-23.

[8] Liu, D., Marcus, A., Poshyvanyk, D., and Rajlich, V., "Feature Location via Information Retrieval based Filtering of a Single Scenario Execution Trace", in Proc. of International Conference on Automated Software Engineering, Atlanta, Georgia, Nov. 5-9 2007, pp. 234-243.

[9] Marcus, A., Sergeyev, A., Rajlich, V., and Maletic, J., "An Information Retrieval Approach to Concept Location in Source Code", in Proc. of Working Conference on Reverse Engineering, Delft, The Netherlands, Nov. 9-12 2004, pp. 214-223.

[10] Poshyvanyk, D., Guéhéneuc, Y. G., Marcus, A., Antoniol, G., and Rajlich, V., "Feature Location using Probabilistic Ranking of Methods based on Execution Scenarios and Information Retrieval", IEEE Trans. on Software Engineering, vol. 33, no. 6, June 2007, pp. 420-432.

[11] Poshyvanyk, D., Marcus, A., and Dong, Y., "JIRiSS - an Eclipse plug-in for Source Code Exploration", in Proc. of International Conference on Program Comprehension, Athens, Greece, 2006, pp. 252-255.

[12] Poshyvanyk, D., Marcus, A., Dong, Y., and Sergeyev, A., "IRiSS - A Source Code Exploration Tool", in Proc. of IEEE International Conference on Software Maintenance, Budapest, Hungary, Sept. 2005, pp. 69-72.

[13] Poshyvanyk, D., Petrenko, M., Marcus, A., Xie, X., and Liu, D., "Source Code Exploration with Google ", in Proc. of International Conference on Software Maintenance, Philadelphia, PA, 2006, pp. 334 - 338.

[14] Robillard, M. P. and Murphy, G. C., "FEAT: a tool for locating, describing, and analyzing concerns in source code", in Proc. of ICSE'03, Portland, OR, pp. 822-823.

[15] Robillard, M. P. and Murphy, G. C., "Representing concerns in source code", ACM Transactions on Software Engineering and Methodology vol. 16, no. 1, Feb. 2007, pp. 1-38.

[16] Robillard, M. P. and Weigand-Warr, F., "ConcernMapper: Simple View-Based Separation of Scattered Concerns", in Proc. of Eclipse Technology Exchange at OOPSLA, 2005, pp. 65-69.

[17] Shepherd, D., Fry, Z., Gibson, E., Pollock, L., and Vijay-Shanker, K., "Using Natural Language Program Analysis to Locate and Understand Action-Oriented Concerns", in Proc. of International Conference on Aspect Oriented Software Development, 2007, pp. 212-224.

[18] Shonle, M., Neddenriep, J., and Griswold, W., "AspectBrowser for Eclipse: a case-study in plug-in retargeting", in Proc. of OOPSLA Workshop on Eclipse Technology eXchange, 2004, pp. 78-82.

[19] Weigand-Warr, F. and Robillard, M. P., "Suade: Topology-Based Searches for Software Investigation", in Proc. of ICSE'08, May 2008, pp. 780-783.

[20] Wilde, N. and Scully, M., "Software Reconnaissance: Mapping Program Features to Code", Software Maintenance: Research and Practice, vol. 7, Jan.-Feb. 1995, pp. 49-62.