Strathcona Example Recommendation Tool

Reid Holmes & Robert J. Walker
Department of Computer Science
University of Calgary
2500 University Dr. NW
Calgary AB Canada T2N 1N4
rtholmes, rwalker@cpsc.ucalgary.ca

Gail C. Murphy
Department of Computer Science
University of British Columbia
201-2366 Main Mall
Vancouver BC Canada V6T 1Z4
murphy@cs.ubc.ca

ABSTRACT

Using the application programming interfaces (API) of large software systems requires developers to understand details about the interfaces that are often not explicitly defined. However, documentation about the API is often incomplete or out of date. Existing systems that make use of the API provide a form of implicit information on how to use that code. Manually searching through existing projects to find relevant source code is tedious and time consuming. We have created the Strathcona Example Recommendation Tool to assist developers in finding relevant fragments of code, or examples, of an API's use. These examples can be used by developers to provide insight on how they are supposed to interact with the API.

Categories and Subject Descriptors

D.2.3 [**Coding Tools and Techniques**]: Program Editors; D.2.3 [**Coding Tools and Techniques**]: Object-Oriented Programming

General Terms

Languages

Keywords

recommender, examples, software structure

1. INTRODUCTION

Learning how to use complex object-oriented frameworks is a difficult task. Frameworks often provide four sources of information that a developer might use to understand such properties: (1) high-level documentation; (2) code-level documentation, such as API details; (3) hand-crafted examples that illustrate how to use specific parts of the framework; and (4) the source code of the framework itself, including embedded comments. Unfortunately, developers often find that documentation and hand-crafted examples are sparse in content, completely absent, or out-of-date [6]. One of the best ways to understand a framework is to see it in use [4].

Ideally, a developer who is new to using a framework could approach one of their experienced colleagues with questions about how to use the framework appropriately. This experienced colleague, who is often constrained for time, is liable to provide some code, or a pointer to relevant code inside a larger system they have written before, for the other developer to explore. If a colleague is not readily available, the developer would need to both locate the source code for another project that uses the framework to explore and search it, through lexical searches, in order to find code that could help them with their task. The developer would decide what to search for by looking at any high-level and code-level documentation that exists and by iteratively looking at the results returned by their queries. However, this is a time consuming and error prone task and it can be difficult to locate relevant API usages for a complex framework.

We have created a tool that aims to help developers quickly and easily access the framework usage information embedded in existing projects. Our tool, Strathcona, leverages the structural characteristics of both the past projects and the developers current context to automatically recommend relevant examples. These examples can be navigated visually and textually; applicable source code from the examples extracted from the existing code can be integrated into the developers current project.

The Strathcona prototype has been implemented as a plug-in for the Eclipse Integrated Development Environment.[1] Although the tool itself is made for Eclipse, it can be used as a development aid for any framework that has had a repository populated for it. Repositories are easy to create as the tool can automatically extract all of the necessary structural information from past projects by traversing their source code. Our ICSE 2005 paper [3] highlights the primary contribution of our approach: the method by which we match the developer's context to the repository to find the most relevant examples. Queries to the repository are automatically generated from structural details of the developer's current development activity; examples are then matched and returned for further investigation. As a result, finding source code examples relevant to the developer's current task costs the developer little effort when the Strathcona tool is used.

We have made several improvements to Strathcona since

[1] http://eclipse.org

ESEC-FSE'05, September 5–9, 2005, Lisbon, Portugal.

our ICSE paper was published. Strathcona now includes a tool that automates the addition of new projects to the example repository. Multiple frameworks are supported, and we have created example repositories for Eclipse, JHotDraw[2], and HttpClient[3] . Earlier versions of Strathcona were hampered by usability issues that have now been addressed. This first public demo of Strathcona highlights its features and the tasks for which it was designed to assist developers.

A short description of our approach is given in Section 2. Section 3 describes a scenario that demonstrates how Strathcona forms queries on an example repository, finds relevant examples, and presents them to the developer. Related work is covered in Section 4 and Section 5 concludes.

2. THE APPROACH

Strathcona helps developers by automatically locating source code examples that are relevant to their work. We have developed a series of heuristics to match software artifacts inspired by the way a developer might try to search for them. The process is as follows. (1) A query is generated based on the structural context of the developer's environment. (2) This query is sent to the example repository (typically on another machine) and is compared against a corpus of existing projects. (3) The most structurally relevant examples are returned to the developer. (4) These examples can then be navigated, both graphically and textually, to determine which are most applicable to the current task. (5) The developer can then integrate any source code from the example into their project.

Our prototype is implemented as a client/server system allowing a team of developers to access a single example repository, centralizing the management of the repository for the team. In the remainder of this section we will describe how the query is formed, outline the role of the server and how it is populated with examples, and describe how examples are navigated and utilized.

2.1 Forming the Query

One of our goals when designing Strathcona was to minimize the amount of effort required for a developer to interact with the tool. The first step in this process was to automatically generate the query so the developer would not need to learn any query languages or have any knowledge of how the repository is organized. The query is initiated by the developer by selecting a program element and selecting the query option; currently, Strathcona supports classes and methods as query selections. The query is based exclusively on the *structural context* of the source code selected by the developer. For a class, its structural characteristics include its type, the types of its parents, and the types of its fields. For a method, its structural characteristics include its signature and the message sends and object instantiations it invokes. The names of fields, methods, and their parameters are not used as these are not structural and are subject to the naming problems that hinder other text-matching approaches. This structural information is sent to the example repository for analysis.

Typically, the developer would write some simple code based on their investigation of the frameworks documentation, API, or code. The code doesn't need to compile in an error free manner (for example, you can write code that calls a method on an interface). Often initial queries are used solely as an investigation tool to examine different framework APIs. As the developer gains a better understanding of the APIs, and their constraints, their context becomes more complete and the examples that are returned by Strathcona help with more fine-grain interactions or details such as error handling.

[2]http://jhotdraw.sf.net

[3]http://jakarta.apache.org/commons/httpclient/

2.2 Locating Examples

Structurally relevant examples are located within the repository by comparing the structural context of the query against the structural context of the classes and methods within the repository. Queries generally take less than ten seconds to return to the developer. We have developed six heuristics that match the context against the repository [3]. These heuristics are based on the inheritance hierarchy, field types, method calls, and object usages. The structural contexts of the ten most relevant examples are returned to the developer for their perusal.

2.3 Navigating Relevant Examples

The developer can then navigate through the returned examples to determine which are most relevant to the task at hand. These examples are displayed graphically by default, with the option to view both the rationale explaining why they were selected as well as their source code. Once a relevant example has been identified, the developer can integrate any of its source code into their current development environment. If necessary, the repository can be queried again after more code has been written as the developer may have a better idea of which parts of the framework they are interested in.

2.4 The Example Repository

The example repository can be automatically generated from existing source code. There is no minimal set of code required to create a repository, although the size of the code base will affect the diversity of the returned examples. The source code is parsed and its structural relationships added to the repository database. Our existing example repositories range in size from 1,000 classes to over 25,000 classes. Developers who want to add projects to an existing repository, or create a new repository, can use another tool we have created to automatically extract the source structure and code from the project into a zip file. This file can be sent to the repository maintainer who can automatically add the project to an existing repository or seed a new one with it.

3. MOTIVATING EXAMPLE

This example highlights how the Strathcona Example Repository may be used by a developer to complete a task. The developer has been asked to evolve their current system to use a third party library, called HttpClient, for handling `http` connections from their current home-grown solution. The system must be able to use `http POST` to send an XML file to a remote server.

The developer started by looking at the documentation for HttpClient. Through the documentation they learned that they would need to use the `HttpClient` and `PostMethod` classes. The fragment shown in the text pane in Figure 3

shows the results of this investigation, along with some of the developer's existing code for parsing an XML file which they took from their previous implementation. Based on this context the developer queried the example repository.

The visual representation of one of the ten returned examples is shown at the bottom of Figure 3. This example seemed applicable and by looking at the code the developer was able to see that it dealt with their exact task. From this example the developer also learned that parsing the XML file was unnecessary as the file was just treated as a stream. Upon copying code from the example, and removing the extraneous parsing code, the developer was able to complete the task. The developer then checked other returned examples and found that one showed a better way of checking the response code; they integrated this code into their environment as well. The final result is shown in Figure 2. By using an example the developer did not waste time trying to find out how to integrate a parsed XML document with `HttpClient`; they also learned how to set the content-type for the `POST` and how to properly handle error conditions.

```
private void sendXMLDocument() {
  HttpClient client = new HttpClient();
  PostMethod post = new PostMethod("http://foo.com");
  String fName = "xmlFile.xml";
  String contentType = "text/xml; charset=ISO-8859-1";

  File xmlFile = new File(fName);

  post.setRequestEntity(new InputStreamRequestEntity(
  new FileInputStream(xmlFile), xmlFile.length()));
  post.setRequestHeader("Content-type", contentType);
  try {
    client.executeMethod(post);

    if (post.getStatusCode() == HttpStatus.SC_OK)
System.out.println(fName+" posted");
    else
System.out.println("Error: "+post.getStatusLine());
  } catch (HttpException e) {
    e.printStackTrace();
  } catch (IOException e) {
    e.printStackTrace();
  } finally {
    post.releaseConnection();
  }
}
```

Figure 2: Initial Method Stub with Integrated Example Code

This simple demonstration illustrates shows how a relevant example was found by matching on only two object instantiations (`HttpClient` and `PostMethod`). Strathcona found for the developer not only examples of the desired functionality, but also examples of related functionality (setting the content type and handling the response code) that the developer would not otherwise have recognized as relevant.

4. RELATED WORK

Several past projects have aimed to help developers locate source code examples. These approaches have relied on a variety of methods to help locate relevant examples.

CodeBroker [8, 7] is the most similar tool to Strathcona. CodeBroker automatically searches a repository of code using comments the developer has entered into the code. CodeBroker also compares method signatures from the code to the repository in order to refine its results. CodeBroker actively queries the repository without developer intervention to keep the results as up to date as possible. The primary drawback with the CodeBroker approach is that it relies heavily on comments, both in the example code and in the code being written by the developer.

CodeFinder [2] uses a query browser to help the developer construct queries that can then be sent to the repository. This helps the developer make more effective queries to the repository as the tool is able to craft the query in a way that can be best used by the repository. By constructing the queries automatically Strathcona avoids this query construction phase and at the same time manages to send the most optimal query to the repository for matching.

Another tool that automatically builds queries to send to the repository is the Hipikat tool [1]. Hipikat creates links between different sources of information in a project, including source files, cvs commits, bug reports, newsgroup postings, and web articles. Developers can find related documents by selecting query on the currently open document. Strathcona takes a more specific approach than Hipikat by supporting only code-level entities in a more fine grain fashion.

The Reuse View Matcher [5] uses a repository of constructed examples to demonstrate how individual classes in a framework are to be used. As these example are constructed by the framework authors their correctness and applicability can be more assured. However, this approach suffers from the fact that creating the examples is time consuming and their coverage cannot be complete. As Strathcona can automatically extract the structure of existing code to create examples it can overcome the time burden required to create a repository.

The primary advantages of Strathcona compared to past projects is the automatic query generation and repository generation. This allows minimal developer effort to query the server or create an example repository. Only CodeBroker has these same traits and this approach relies heavily on comments in the code, which developers often find to be absent, incomplete, or inaccurate [6]. By relying on the structure of the software we ensure that our results remain current with respect to how the system is actually being used.

4.1 Drawbacks of our Approach

This approach is not perfect. The primary problem with this approach is that the developer must have some idea of where to start before they can write any code to query Strathcona with. This initial insight would hopefully be supported by some type of documentation or code exploring. We have performed a case study to ensure that Strathcona can return relevant results for developers working on pre-selected tasks [3]. Preliminary informal evaluation at the University of Calgary (conducted after the ICSE paper) indicates that undergraduate students were often able to identify a starting point for a query, although the factors determining quality of this starting point have not been determined.

The effectiveness of the approach also depends on comprehensiveness of the example repository. By providing support for the automatic analysis and addition of projects to the repository we aim to ease the set-up process for the tool. In order to prevent low-quality examples from being returned to developers the repository maintainer needs to make a judgement call when deciding which projects to add

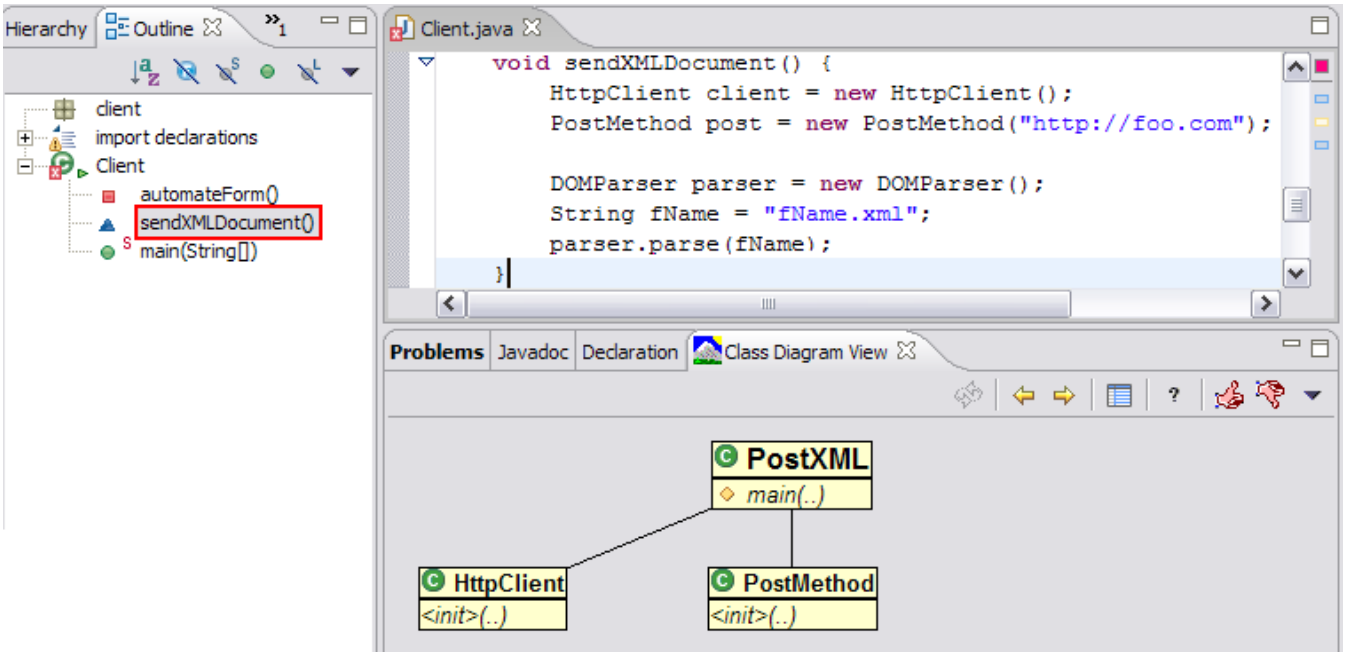

Figure 1: Strathcona: Showing a returned example and the code snippet that led to its selection

to the repository. One potential solution is to only add the framework itself, along with code produced by the framework developers. This is the approach we have used for our Eclipse example repository.

5. CONCLUSION AND FUTURE WORK

The Strathcona tool has been created to bridge the gap between source code and documentation. Existing projects that make use of a framework implicitly document how the framework can be used. By providing a low-cost querying mechanism on an example repository that can be populated automatically from existing source code, we have created a tool that can help the developer learn how use a framework. Currently we have populated three repositories with systems ranging from 10KLOC to over 2MLOC. We have successfully used the Strathcona tool to complete tasks using each of these repositories. By providing developers with an easy mechanism to search a large corpus of previously written code we hope to reduce their development burden by leveraging past code that has been written, tested, and deployed.

Our initial informal evaluations into the difficulty of creating blocks of code to query from needs to be expanded to determine what factors of the query affect the quality of the results. Going forward we hope to use the tool to support the developer reuse examples instead of just using them to highlight how a framework is to be used.

6. REFERENCES

[1] D. Cubranic and G. C. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Proceedings of the 25th International Conference on Software Engineering*, pages 408–418. IEEE Computer Society, 2003.

[2] S. Henninger. Retrieving software objects in an example-based programming environment. In *Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 251–260. ACM Press, 1991.

[3] R. Holmes and G. C. Murphy. Using structural context to recommend source code examples. In *ICSE '05: Proceedings of the 25th International Conference on Software Engineering*, New York, NY, USA, 2005. ACM Press.

[4] R. E. Johnson. Documenting frameworks using patterns. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 63–72, 1992.

[5] M. B. Rosson and J. M. Carroll. The reuse of uses in Smalltalk programming. *ACM Transactions on Computer-Human Interaction*, 3(3):219–253, 1996.

[6] J. Singer. Practices of software maintenance. In *International Conference on Software Maintenance*, pages 139–145, 1998.

[7] Y. Ye and G. Fischer. Supporting reuse by delivering task-relevant and personalized information. In *Proceedings of the 24th International Conference on Software Engineering*, pages 513–523. ACM Press, 2002.

[8] Y. Ye, G. Fischer, and B. Reeves. Integrating active information delivery and reuse repository systems. In *Proceedings of the 8th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 60–68. ACM Press, 2000.