CodeGenie: a Tool for Test-Driven Source Code Search

Otávio Augusto Lazzarini Lemos
Departamento de Sistemas de Computação,
ICMC/USP - São Carlos - Caixa Postal 668
13560-970 São Carlos-SP-Brasil
oall@icmc.usp.br

Sushil Bajracharya and Joel Ossher
Donald Bren School of Information and
Computer Sciences
University of California, Irvine
{sbajrach,jossher}@ics.uci.edu

Abstract

We present CodeGenie, a tool that implements a test-driven approach to search and reuse of code available on large-scale code repositories. With CodeGenie, developers design test cases for a desired feature first, similar to Test-driven Development (TDD). However, instead of implementing the feature from scratch, CodeGenie automatically searches for an existing implementation based on information available in the tests. To check the suitability of the candidate results in the local context, each result is automatically woven into the developer's project and tested using the original tests. The developer can then reuse the most suitable result. Later, reused code can also be unwoven from the project as wished. For the code searching and wrapping facilities, CodeGenie relies on Sourcerer, an Internet-scale source code infrastructure that we have developed.

Categories and Subject Descriptors D.2.13 [*Software Engineering*]: Reusable Software

General Terms Languages

Keywords source code search, source code reuse, test-driven development, test-first

1. Introduction

Large scale availability of Open Source code in the Internet has opened up new possibilities for code search and reuse. However, the process of formulating code queries, filtering the results and weaving them in a developer's workspace is still painstakingly laborious.

Recently there has been some work in developing search engines specifically targeted at source code [1, 2, 3]. While these systems are promising, they do not leverage the various complex relations present in the code, and therefore have limited features and search effectiveness. In particular: (1) there is no strong support for integration of these search facilities in a development environment; (2) the mechanisms for expressing code queries are usually limited to keywords; and (3) there is little guarantee that the retrieved results correctly implement the behavior of the desired function and whether it would function properly in the local context.

Concerned with these limitations, and based on Sourcerer – a source code infrastructure developed within our research group [4, 5] – we are currently investigating a new approach to source code search and reuse that integrates the use of test cases as inputs for the code search queries. We call this approach Test-Driven Code Search (TDCS). In TDCS, test cases serve two purposes: (1) they define the behavior of the desired feature to be searched; and (2) they test the matching results for suitability in the local context.

2. Test-Driven Code Search

The same way that test cases can be used to define a software feature in Test-Driven Development (TDD) [6], they can also be used to describe a desired feature in a code search task. Moreover we can take advantage of other characteristics of TDD, in this context: (1) Feedback – test cases provide instant feedback about the suitability of a particular code result in the local context; (2) Task-orientation – the requirement of designing test cases first guides the developer in searching for self-contained and manageable software pieces, one at a time; and (3) Quality assurance – since code results might come from many different sources, tests cases help in assuring its quality.

With TDCS, cycles similar to Test-First cycles can be followed: (1) construct test cases; (2) complete the application by integrating code search results; and (3) refactor, if necessary. Thus, instead of always implementing the function as in TDD, the developer can search for it first.

Figure 1 shows a basic TDCS process. To describe a missing feature in the project, test cases are designed in the Integrated Development Environment (IDE). The search facility can then be triggered and, based on the information available in the test cases, a query is sent to a code search service capable of processing it. In the IDE, the developer

Copyright is held by the author/owner(s).
OOPSLA'07, October 21–25, 2007, Montréal, Québec, Canada.
ACM 978-1-59593-865-7/07/0010.

can explore the results by weaving/testing and unweaving them. To do that, a program slicing service to provide self-contained code pieces related to the desired feature, and a repository access service must be available at the Code Services side. Whenever the developer feels satisfied with a particular code result, he/she can leave it woven to the project. Unweave of a code result can also be done.

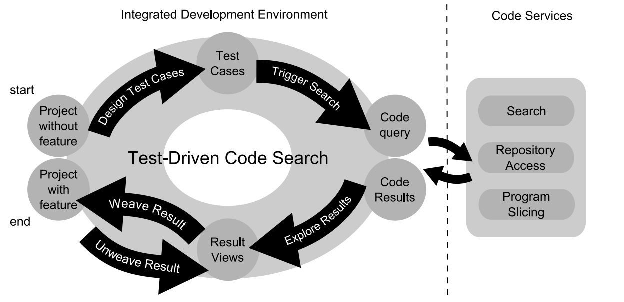

Figure 1. TDCS process.

3. CodeGenie: a TDCS implementation

The IDE side of our TDCS implementation uses Eclipse, an extensible platform for tool integration that provides several Java software development services. We believe that Eclipse is suitable because its extensible nature makes it easy to integrate the particular TDCS features.

The test case design part of CodeGenie[1] is supported by JUnit, which is fully integrated with Eclipse. JUnit test classes must be created to define the desired features. The current entry point for a search task in CodeGenie is a single method, thus test cases have to target at least one missing method (which may be inside an existing or non-existing class in the current application).

Once the test class is created, CodeGenie is ready to extract information about the desired feature. The tool extracts the interface of the missing method, and the names of the missing method and of the class it belongs to. It does that by analyzing the compilation errors present in the test cases due to the missing method or class. The Abstract Syntax Tree (AST) of the test class is explored to extract the return type and argument types of the missing method.

For the weaving and unweaving of code results, CodeGenie applies the merge by name strategy as used in Hyper/J. It simply copies all classes inside the chosen code result into the developer's project and merges classes with coincident names. The merging is done by a union operation on the structures of the classes. If there are coincident methods or fields, the structures that were already present in the code have priority over the ones being added.

To track the woven structures inside the developer's project CodeGenie uses Java annotations. Every code structure that is woven to the developer's project is annotated with a `@FromSlice` annotation indicating from which code result it comes from. A `name` element is used to identify the slice. In this way when a unweave operation is triggered, CodeGenie removes all code structures coming from the related slice by analyzing the annotations.

[1] `http://sourcerer.ics.uci.edu/codegenie`

To test the woven project we use the JUnit functionality that comes with Eclipse. CodeGenie communicates with this plugin so that when search results are woven and tested, the testing results are updated in the CodeGenie Search View.

The Code Services part of CodeGenie is implemented by Sourcerer [5]. Sourcerer provides three main facilities: (1) Code Search, which implements a code query processing facility to clients; (2) Repository Access, which provides access to the managed repository of code in Sourcerer; and (3) Slicing, which provides a slicing facility that generates self-contained pieces of code based on specific entry points.

4. Conclusion

CodeGenie is the first implementation of a TDCS tool and it provides evidence that TDCS is feasible. We also used CodeGenie to search for several types of features and found reusable implementations for all of them. Table 1 shows 14 examples, including information about the number of found candidates, the number of runnable candidates, and the number of working candidates. These initial results suggest that TDCS is an effective way to enable reuse of code from shared repositories.

Table 1. Some feature searches using CodeGenie.

Feature	total	# runnable	# working
arabic to roman	6	6	2
arabic to ordinals	8	8	1
arabic to alpha	6	6	1
quicksort	4	2	2
complementary DNA	3	3	1
reverse complementary DNA	2	2	2
filtering folder contents	1	1	1
unzip	7	5	3
zip	5	5	2
hexadecimal to strings	6	4	2
strings to hexadecimal	6	4	2
byte arrays to hexadecimal	2	2	2
hyphenated to camel case	2	2	1
camel case to phrases	3	3	1

References

[1] Koders web site. `http://www.koders.com`.

[2] Krugle web site. `http://www.krugle.com`.

[3] Google Code Search. `http://www.google.com/codesearch`.

[4] Sourcerer web site. `http://sourcerer.ics.uci.edu`.

[5] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, and C. Lopes. Sourcerer: a search engine for open source code supporting structure-based search. In *OOPSLA '06: Companion to the 21st ACM SIGPLAN OOPSLA*, pages 681–682, New York, NY, USA, 2006. ACM Press.

[6] K. Beck. *Test Driven Development: By Example*. Addison-Wesley Professional, November 2002.