The Journal of Systems and Software 130 (2017) 81–93

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A search engine for finding and reusing architecturally significant code

Ibrahim Jameel Mujhid, Joanna C. S. Santos, Raghuram Gopalakrishnan, Mehdi Mirakhorli*

Software Engineering Department, Rochester Institute of Technology, Rochester, NY, United States

A R T I C L E I N F O

Article history:
Received 16 February 2016
Revised 12 August 2016
Accepted 23 November 2016
Available online 29 November 2016

Keywords:
Architecture
Traceability
Tactics
Information
Models

A B S T R A C T

Architectural tactics are the building blocks of software architecture. They describe solutions for addressing specific quality concerns, and are prevalent across many software systems. Once a decision is made to utilize a tactic, the developer must generate a concrete plan for writing code and implementing the tactic. Unfortunately, this is a non-trivial task even for experienced developers. Often, developers resort to using search engines, crowd-sourcing websites, or discussion forums to find sample code snippets to implement a tactic. A fundamental problem of finding implementation for architectural tactics/patterns is the mismatch between the high-level intent reflected in the descriptions of these patterns and the low-level implementation details of them. To reduce this mismatch, we created a novel Tactic Search Engine called ArchEngine (ARCHitecture search ENGINE). ArchEngine can replace this manual internet-based search process and help developers find and reuse tactical code from a wide range of open source systems. ArchEngine helps developers find implementation examples of an architectural tactic for a given technical context. It uses information retrieval and program analysis techniques to retrieve applications that implement these design concepts. Furthermore, it lists and rank the code snippets where the patterns/tactics are located. Our case study with 21 graduate students (with experience level of junior software developers) shows that ArchEngine is more effective than other search engines (e.g., Krugle and Koders) in helping programmers to quickly find implementations of architectural tactics/patterns.

1. Introduction

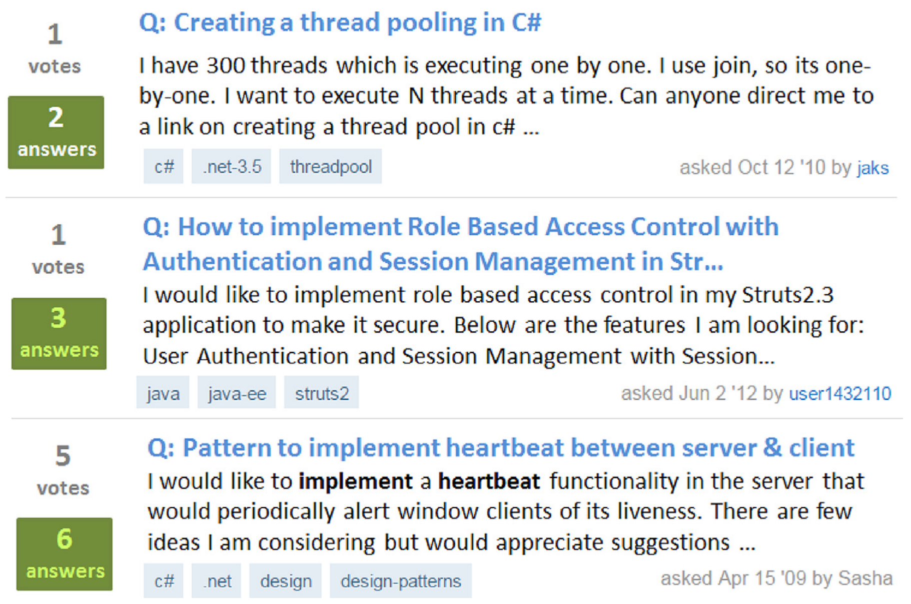
In order to speed up the development process, many programmers reuse existing code. Often they find that there are generic functionalities that other programmers wrote and these fragments can be reused. Socio-technical websites such as StackOverflow, and code search engines (e.g., Google Code, Koders) are the primary resources that developers often use for finding and reusing source code or even for getting ideas on how to implement a feature. However, this can be challenging when it comes to reusing *architecturally significant codes* (Mirakhorli et al., 2012b) – code snippets that implement architectural patterns (Hanmer, 2007) and tactics (Bass et al., 2003). A fundamental problem is related to the difficulties in identifying and tagging architectural patterns and tactics in the source code of a project. As a result, the current search engines such as Google Code, Koders or even those developed in academia (Mcmillan et al., 2013) fail to incorporate these design concepts in their underlying search algorithms.

With the increasingly adoption of iterative incremental software development practices and integration of coding and design activities, there is a growing need for search engines that helps developers identify and reuse code snippets related to the architectural patterns/tactics. In a simple search through the web, one can find several examples of online posts made by developers requesting help in online forums because they did not understand how to implement specific patterns/tactics. Fig. 1 shows three examples of such questions. One developer is seeking help regarding the generic implementation of a *Pooling* tactic in C#. While two others are looking for specific implementation of tactics in particular context/technology. Another one wants to implement *role-based access control* along with *Struts framework*, while the third one is seeking samples to implement *heartbeat* reliability tactic between *clients and a server*.

These examples show that typically developers' query for a sample tactical code has two parts, (i) the desired **tactic** and (ii) a particular **context** or **technology** in which the tactic needs to be implemented. Therefore, a search engine not only needs to identify and index occurrence of architectural tactics, but also needs to identify the **technical context** in which the tactic is implemented. State of the art, in the area of enhancing code reuse, relies on application of data-mining and natural language processing (NLP) techniques to build source code recommender sys-

* Corresponding author.
E-mail addresses: ijm9654@rit.edu (I.J. Mujhid), jds5109@rit.edu (J.C. S. Santos), rg8772@rit.edu (R. Gopalakrishnan), mehdi@se.rit.edu, mxmvse@rit.edu (M. Mirakhorli).

http://dx.doi.org/10.1016/j.jss.2016.11.034
0164-1212/© 2016 Published by Elsevier Inc.


1 votes
2 answers
Q: Creating a thread pooling in C#
I have 300 threads which is executing one by one. I use join, so its one-by-one. I want to execute N threads at a time. Can anyone direct me to a link on creating a thread pool in c# ...
c# .net-3.5 threadpool
asked Oct 12 '10 by jaks

1 votes
3 answers
Q: How to implement Role Based Access Control with Authentication and Session Management in Str...
I would like to implement role based access control in my Struts2.3 application to make it secure. Below are the features I am looking for: User Authentication and Session Management with Session...
java java-ee struts2
asked Jun 2 '12 by user1432110

5 votes
6 answers
Q: Pattern to implement heartbeat between server & client
I would like to **implement** a **heartbeat** functionality in the server that would periodically alert window clients of its liveness. There are few ideas I am considering but would appreciate suggestions ...
c# .net design design-patterns
asked Apr 15 '09 by Sasha

Fig. 1. Developers seek help in online forums to implement architectural patterns/tactics.

tems (McMillan et al., 2012b), and search engines (Chatterjee et al., 2009a; Grechanik et al., 2010; Stylos and Myers, 2006). However, the primary intent of these techniques is retrieving generic functional code and not tactical code. Moreover, these techniques do not differentiate between the search concept and the technical context where the concept is implemented.

The difficulty of detecting architectural patterns/tactics as well as challenges for identifying the technical context within the source code of a project are the two main reasons that the notion of reusing architecturally significant source code is not well explored in the software architecture community.

In this paper, we address these limitations by presenting and rigorously validating ArchEngine, a novel search engine designed for retrieving architecturally significant code snippets. ArchEngine supports the developers in finding and reusing relevant source files that implement an architectural tactic using a particular technology. The users search request is reflected in query terms (i.e., *Authentication* using *Spring Framework* over *HTTP* channel).

To build a source code search engine for architectural tactics, ArchEngine: (i) relies on a novel text-based classification technique to automate the discovery, extraction and indexing of architectural tactics across 116,609 open-source systems. (ii) It implements a big data compatible architecture to search efficiently through 22 million source files of these projects. (iii) It uses information retrieval (Manning et al., 2008b) and structural analysis techniques (Pressman, 2005; Mathiassen et al., 2000) to detect tactics and to identify the technical context in which the tactic has been used. Lastly, (iv) it utilizes a novel ranking algorithm to order the retrieved tactical files based on both tactic-correctness and relevancy to the technical context stated in the users query.

In our case study, 21 graduate students with the industry experience level equivalent to a junior software developer[1] evaluated the accuracy and practicality of the ArchEngine. The results show, with strong statistical significance, that users find more relevant tactical code snippets with higher precision when they use ArchEngine rather than other search engines such as Krugle, Koders, GitHub, and Portfolio. ArchEngine is available for public use at [2].

[1] Between one year to three years of software development experience.
[2] http://design.se.rit.edu/ArchEngine/.

The remainder of this paper is structured as follows. Section 2 provides an overview of the approach. Sections 3 and 4 describe the process of mining and indexing the source code of 116,609 open source projects. Sections 5 and 6 describe detection of architectural tactics and implementation/technical context in open source projects. Sections 7 and 8 describe the ranking algorithms used to sort the results and the search process, respectively. Section 9 represent the empirical experiments that were conducted to evaluate the search engine. Section 10 describes the related work, Section 11 explains the threats to validity of this work and Section 12 summarizes the contributions of this paper and discusses future work.

2. Overview of approach

The architecture of our search engine and its components are depicted in Fig. 2. The first component is an *ultra-large-scale source code repository*, which contains over 116,609 open-source projects extracted from various online software repositories. The second component is our novel *source code indexing* technique, which represent projects and their source files in a form of index that is efficient for performing information retrieval techniques. The third component is a *tactic detector* (Mehdi Mirakhorli, 2016; Mirakhorli et al., 2012b) which is capable of detecting various architectural tactics in the indexed code artifacts. The tactic detector relies on information retrieval techniques, and its accuracy was previously validated in a series of experiments (Mehdi Mirakhorli, 2016; Mirakhorli et al., 2012b).

The fourth component is a *dependency analyzer*, which generates a dependency matrix for each tactical file in the source code of a project. This matrix is then used by the fifth component – *Matching Technical Problem* – to find whether the implementation of a given tactic is related to a technical problem/context or not. Technical context refers to a framework, technology, programming language, or API which can be used to implement the tactic. In other words, it is the technical problem in which the tactic needs to be implemented.

The final component is a novel *Ranking algorithm*. It ranks the source files in the search results based on (i) the semantic similarity of a source file to a searched tactic (ii) the semantic similarity