Facilitating the Comparison of Software Retrieval Systems through a Reference Reuse Collection

Oliver Hummel
Software Engineering Group
University of Mannheim
68131 Mannheim, Germany
Phone: +49 6 21 / 1 81 - 39 29
hummel@informatik.uni-mannheim.de

ABSTRACT

Although the idea of component-based software reuse has been around for more than four decades the technology for retrieving reusable software artefacts has grown out of its infancy only recently. After about 30 years of basic research in which scientists struggled to get their hands on meaningful numbers of reusable artifacts to evaluate their prototypes, the "open source revolution" has made software reuse a serious practical possibility. Millions of reusable files have become freely available and more sophisticated retrieval tools have emerged providing better ways of searching among them. However, while the development of such systems has made considerable progress, their evaluation is still largely driven by proprietary approaches which are all too often neither comprehensive nor comparable to one another. Hence, in this position paper, we propose the compilation of a reference collection of reusable artifacts in order to facilitate the future evaluation and comparison of software retrieval tools.

Categories and Subject Descriptors: H.3.7 [**Information Storage and Retrieval**]: Digital Libraries – *standards.*

General Terms: Measurement, Standardization.

Keywords: Component-based software development, information retrieval, reference reuse collection.

1. INTRODUCTION

Mainly triggered by the "open source revolution", the research effort spent on the retrieval of reusable software artifacts has experienced a tremendous boost in recent years. Although software reuse was identified as a promising approach to overcome the "software crisis" over four decades ago [1], and a number of seminal publications (such as [10] or [7]) delivered important groundwork for current software retrieval systems, reuse research struggled to produce practically usable results, as e.g. effective repository systems and integrated CASE tools to use them. Previous research identified a lack of reusable artifacts as one of the main reasons for this dilemma and proposed to overcome this by crawling the WWW and the repositories of open source hosting sites for reusable software assets [16]. This idea of internet-scale software search engines clearly spread not only in the research community that developed search engines such as Spars-J [4], Merobase [8], or Sourcerer [11], but in industry as well (see e.g. the code search engines of Google, Koders or Krugle). Today, there are at least a dozen software search engines available on the web. They mostly allow users to search for reusable source files based upon retrieval algorithms of different sophistication (see e.g. [8] for a more comprehensive overview). But not only the back-end search functionality improved considerably in recent years, but also the user front ends. While a web-based "google-style" search interface is perhaps sufficient for occasional users, Ye [14] was amongst the first researchers that realized it would be more effective if software developers had proactive tool support directly in the IDEs they are using for their development work. His so-called CodeBroker was the pioneering tool that monitored the activities of a developer and automatically proposed reuse candidates that it considered appropriate. However, CodeBroker was only based on a rather small reuse collection. Only Hummel et al.'s Code Conjurer tool [8] recently integrated powerful retrieval algorithms, a large component collection and a proactive recommendation engine into the widespread Eclipse IDE. Other noteworthy progress includes the work of Inoue et al. [4] that adapted Google's webpage-based Pagerank algorithm to software retrieval by prioritizing search results according to the frequency they are used by other artifacts.

However, although all these approaches are certainly a step in the right direction and brought new and interesting ideas into the community they all share one significant problem. To date, evaluations of these tools are largely based on proprietary data and thus there is currently no way to compare their results, making it hard for researchers to give clear recommendations to practitioners that might contemplate the use of such a tool. As for example stressed by Basili [2]: *"Proposing a model or building a tool is not enough. There must be some way of validating that the model or tool is an advance over current models or tools"*. Interestingly, this is a problem that is or has been shared by other communities as well. First and foremost, it is clearly the information retrieval community [12], which is obviously closely related with component retrieval anyway, that was experiencing similar problems. In the early years of this community there were also a lot of new and exiting ideas as well as prototypes around, but the proprietary (and often expensive) evaluations performed on them were usually not very helpful and especially not comparable with each other. However, this community was able to overcome this challenge by defining so-called reference collections basically comprising a large collection of documents, a number of challenges for retrieval systems and the expected solutions for them (e.g. [15]). The second community that is struggling with the comparability of its tools is the rather young community trying to retrieve and orchestrate (semantic) web services. It has been trying to compare the systems of their contributors by organizing challenges (e.g. http://sws-challenge.org) where the tools are supposed to solve a given exercise by orchestrating a number of services into a new service. This is another interesting idea that we shall pick-up again later in this paper.

1.1 Overview

Given the open issues discussed above, the central theme of this position paper is to propose the establishment of a reference reuse collection that is intended to offer researchers a standard to evaluate their retrieval systems. In our view this will bring two significant advantages - first it will simplify the evaluation of new tools for individual researchers since it will no longer be necessary for them to gather up an own collection and, second, it of course facilitates the direct comparison of approaches and tools. The remainder of this paper is structured as follows: First we very briefly introduce some foundations from information retrieval that explain how retrieval systems are usually evaluated there before we take a look on the current state of the art in the evaluation of software retrieval systems and the gaps that we have identified. After that we propose to create a reuse reference collection in order to facilitate the evaluation of software retrieval solutions and briefly discuss some research challenges associated with this idea. Finally, we conclude our paper with a brief summary of our contribution.

2. FOUNDATIONS

Since software (component) retrieval is based on ideas from general information retrieval (IR) to a large extent, it makes sense to shed some light on the foundations coming from this area. In IR, so-called recall and precision are accepted as the standard measures for determining the efficiency of retrieval systems. Recall is defined as the proportion of all relevant documents that have been retrieved from a document collection for a given query and precision is the proportion of the retrieved documents that are relevant to the query. A more formal description of these concepts is provided by [12], for example. However, this definition makes one important assumption, namely, that the proportion of relevant documents in the collection is known a priori, a prerequisite which is unfortunately not valid for queries in internet-scale web search engines, for instance. In this context, [12] presents two common criticisms that used to plague information retrieval (IR) research, namely the lack of a solid formal framework and the lack of consistent testbeds and evaluation frameworks. Interestingly, software engineering in general, and software retrieval in particular, are obviously subject to the same criticisms (see e.g. the works of Basili [2] resp. Mili et al. [3]).

Due to the inherent psychological subjectiveness associated with information understanding by humans, the IR community has only acted upon the second problem so far: retrieval approaches (i.e. algorithms and tools) for textual information retrieval are typically compared via so-called reference collections where queries are applied to a well-known collection of documents and the expected results are determined by experts. However, until the so-called TREC (for **T**ext **RE**trieval **C**onference) collection [15] with more than one million documents was established in the early 1990s, experimentation in information retrieval was also dominated by small and proprietary "proof-of-concept" test collections (often involving expensive experiments with humans) for nearly thirty years. As mentioned before, for collections of a significant size it becomes a challenge to identify all relevant documents for a query. This, however, is necessary to determine the quality of the systems under evaluation with the help of recall and precision. Thus, a trick had to be applied for creating the TREC collection since its document base is simply too large to be completely overseen by humans: only the queries for evaluating the systems were thought out by experts, the list of relevant documents was created by selecting only those documents that were actually regarded as being relevant by experts out of the results delivered from various IR systems. With this information (which is clearly not perfect, though) it has become much more effective to compare various information retrieval approaches with one another and to derive recall and precision for them in a comparable way. In turn, this has facilitated the improvement of the IR systems themselves as e.g. reported in [15].

2.1 Application to Component Retrieval

Clearly, it is not a new idea to apply recall and precision to software retrieval systems, this has already been done a long time ago. For example, Mili et al. tried to estimate these values for the various retrieval methods they identified in their well-known survey on the topic [3]. However, as stated by the authors, a software retrieval process typically involves two criteria because a candidate artifact can indeed fulfil the matching condition of one specific retrieval technique, but may not necessarily match a user's relevance criterion. For example, a simple keyword-based search technique might retrieve 20 source files matching the term "customer" but only 2 of them might actually fulfil the user's requirements for a customer component (perhaps the other 18 only have a reference to a customer object etc.) and thereby fulfil his relevance criterion. Obviously, finding a good relevance criterion is another challenge for the evaluation of software retrieval systems. This becomes even clearer when one becomes aware that there are at least five basic software retrieval techniques (and thus different matching conditions) that were identified by Mili et al. From today's point of view we prefer to consider their sixth (so-called topological) retrieval method as an approach for ranking results according to their closeness of match to a given query.

3. PROBLEM STATEMENT

Mature research in software reuse, however, is many years younger than in information retrieval. Thus, it is important to mention again that the notion of relevance is clearly different compared to textual retrieval systems. While the latter focuses on "merely" finding meaningful documents in natural language, the basis for software retrieval are programming languages and their more formalized constructs (such as objects or components). Thus, it is possible to define a much tighter definition of relevance in the context of software reuse. In the optimal case, a component can be considered relevant if it matches all required syntactical (i.e. the signature) as well as the semantic (i.e. the functionality) properties to 100%. However, while syntactic matching is essentially a question of pattern matching, it is not guaranteed that a syntactic match also delivers relevant results in terms of functionality. In contrast, relevance in textual information retrieval does not require an exact syntactic match as there exist various ways to express the same information in natural language. Actually, this is true for software as well, but ultimately, a reusable piece of software will only be relevant to a developer if it fully complies with his initial specification.

In other words, the ultimate relevance criterion for a retrieved software artifact is that it can be deployed and re-used "as is" in a given context without any manual modification or adaptation. Thus, potential adapter creation must rather be part of the retrieval system than another burden for the developer. Unfortunately, the few practical evaluation attempts known from literature so far often did not find a practical means to unambiguously specify when an artifact is relevant and thus confined themselves to check the matching condition of the underlying retrieval algorithm instead of the relevance criterion. Clearly, this also makes it hard if not impossible to replicate the evaluations and thus to compare different retrieval algorithms with each other. Even in the high-profile publication of Inoue et al. [4] the matching condition used is not made explicit, but it seems likely that it was merely the appearance of a specific term in the source code. Admittedly, the clear specification of software systems and components is a challenge that has been plaguing software engineering for many decades and only recently the test-driven development community [5] has found a simple and practically usable solution to overcome it. Their idea of using test cases as a specification for components has been picked up and applied by a number of researchers in a reuse context, recently [6], [9], [11]. This so-called test-driven reuse approach seems to be promising for setting up a reference reuse collection as we will discuss in the next section.

The second central problem that has been bothering researchers in the component retrieval community for a long time was getting a large enough software collection in their hands. Thus, early research in the 1990s was based on small and proprietary collections with merely a few hundred components (see e.g. [7], [14]). Even worse, due to the small number of components indexed in these prototypes, the experimental tasks used for the respective evaluations look very much as if they were (clearly out of necessity) optimized for the contents of the repository. Thus, it is very difficult to judge whether these tools would have received the same impressive appraisals in scaled-up environments containing millions of artifacts. Only very recently, the growing amount of open-source software available on the Internet allowed carrying out experiments with larger collections. For instance, Inoue et al. [4] have experimented on about 150.000 files collected in SparsJ, the Sourcerer search engine used by Lemos et al. e.g. in [11] has collected about 560.000 files and Hummel [8] and Reiss [9] experimented with the help of the search engines Merobase resp. Google Codesearch that each contain millions of indexed artifacts.

4. SOLUTION OUTLINE

As comparability and reproducibility are the tenets of good research [2], it is certainly important that our community joins forces in order to define a reference collection for the evaluation of software retrieval tools and algorithms. At first sight, it looks as if we have all ingredients ready: millions of source files are freely available from the Internet, new technologies are available to better assess the relevance of reusable artifacts and we should be able to use seminal ideas from the information retrieval and web service communities as a basic blueprint for our efforts. Thus, our initial proposal for a reuse reference collection includes indexing a larger number of open source projects in order to establish the base collection. Second, a survey of previous tool evaluations should be carried out in order to identify usable and expressive enough reference examples that can be used within the collection and to create new ones if necessary. Last but not least, clear criteria need to be established when a component can be considered as relevant for a given query. Given the recent experience with test-driven reuse, it seems promising to use test cases as the final relevance criterion, as, to our knowledge, test-driven reuse is the software retrieval technique which comes closest to the demand of being a precise relevance criterion (assuming of course that the test cases are "good enough"). Furthermore, it is even usable with a reasonable amount of effort. However, this clearly is an important decision as the relevance criterion needs to be carefully chosen in order to allow it to be used with any other retrieval approach as well. In [13] we were able to show that it is indeed possible to derive queries for older search techniques (such as keyword or signature matching [3]) from test cases with little effort. Based upon the existing prototypes, it thus seems feasible to identify an initial set of relevant components for each query in the reference collection which could later be extended if better systems should find more reusable candidates. Once this has been accomplished, organizing challenges for retrieval systems similar to the web service community is a logical consecutive step.

4.1 Open Challenges

Query definition, however, is not the only serious question that needs to be addressed; there are a number of other factors that make the creation of a reference collection in software retrieval even more challenging than it was in information retrieval about twenty-five years ago. One important difference between the two areas is the fact that source files can not only be named in various human languages, which is a similar challenge to that in information retrieval, they can also we written in various programming languages for various platforms. Thus, it is an open question whether e.g. Java can be accepted as the "lingua franca" of such a reference collection and the insights gained with it can be easily transferred to other programming languages as well. Even worse, software can appear in source or binary form, whereas the latter is typically much less suited for component retrieval since there is fewer metadata (such as source code or comments) available to facilitate e.g. keyword-based searches. In the extreme case, software might even be delivered as a service where, by definition, no introspection is possible and thus no use of any kind of additional "internal" information is possible.

A second very fundamental issue is the question of what kinds of software should be supported by the reference collection. So far we have implicitly talked about software artifacts that exhibit functionality only via well-defined interfaces. This clearly includes classes and operations in object-oriented languages and (web) services, but it is not yet clear how to deal with and how to specify class assemblies and larger components such as subsystems, for example. Another issue that arises with software is that it can have various dependencies on other artifacts, which means that a component might consist of multiple sub-components or classes and also might require additional components (i.e. libraries) to function. A third significant difference between documents and software is that the latter typically keeps evolving even after a first version has been published. Hence, if we assume that we will pursue the first naïve approach of crawling various open source projects as the starting point for a reference collection, it is an open question whether it should remain static and thus may contain a snapshot with unfinished and faulty files forever or whether it should be updated on a regular basis. The second option will, however, most likely alter the set of relevant components for the queries each time an update is performed. Furthermore, if we allow adaptation of retrieval results it must be asked how much adaptation should be allowed resp. required. In other words, may an adapter simply be a 1:1 wrapper or should it be possible that a façade-style adapter can compose a number of pieces into a larger whole?

Finally, another research question arising is whether the proposed reference collection should focus purely on the retrieval of reusable material as discussed so far? Hummel [13] has identified a basic set of further "usage modes" for software retrieval systems that might be worth supporting, too. For instance, it seems reasonable to use such a system in order to search for missing libraries or to find the source code of a specific open source file more quickly than by browsing the web and checking it out from its version control system. However, to our knowledge there is currently no comprehensive compilation of possible usage modes for software retrieval systems beyond the preliminary overview given by Janjic et al. [17] and consequently it is hard to tell which of them should be supported in a reference collection.

5. CONCLUSION

In this position paper we have explained that the perceived significant improvements made with the development and implementation of component retrieval solutions in recent years, have not yet been backed up by a similar improvement in terms of their evaluation. We have identified two main obstacles that hindered a systematic assessment of retrieval approaches in the past, namely the limited availability of large enough software collections and the difficulty in defining an expressive relevance criterion for retrieved reuse candidates. These two points obviously forced researchers in the past to come up with ad hoc evaluation approaches that were all too often tailored to the retrieval solution they were intended to test. Thereby, the repeatability of evaluations was, and still is, widely limited making the comparison of reuse approaches and recommendations for their practical usage hard if not impossible. Thus, in this paper we have proposed to develop a software retrieval reference collection analogue to the collections built by the information retrieval community when it was faced with similar challenges some twenty years ago.

However, while the idea of setting up a reference collection of reusable components, example queries and expected results is straightforward, the road to its implementation is filled with obstacles. Amongst others, we have especially identified the challenges of finding and formulating meaningful reference queries and relevance criteria as the most important tasks that need to be tackled in order to create a useful reference collection. Nevertheless, should our community be able to overcome these challenges it could benefit considerably from this effort which might help to pave the way towards robust internet-scale component markets as envisaged by McIlroy over forty years ago.

6. REFERENCES

[1] McIlroy, D.: Mass-Produced Software Components, Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 1968.

[2] Basili, V.: The Experimental Paradigm in Software Engineering, LNCS 706, Springer, 1993.

[3] Mili, A., R. Mili and R. Mittermeir: A Survey of Software Reuse Libraries, Annals of Software Engineering 5, 1998.

[4] Inoue, K., R. Yokomori, H. Fujiwara, T. Yamamoto, M. Matsushita, S. Kusumoto.: Ranking Significance of Software Components Based on Use Relations, IEEE Transactions on Software Eng., Vol. 31, No. 3, 2005.

[5] Beck, K. Test-Driven Development by Example, Addison Wesley, 2003.

[6] Hummel, O. and C. Atkinson: Extreme Harvesting: Test Driven Discovery and Reuse of Software Components, Proceedings of the Intern. Conf. on Information Reuse and Integration, 2004.

[7] Podgurski, A., Pierce, L. Retrieving reusable software by sampling behavior. ACM Transactions on Software Engineering and Methodology (Vol. 2, Iss. 3), 1993.

[8] Hummel, O., Janjic, W., Atkinson, C. Code Conjurer: Pulling Reusable Software out of Thin Air, IEEE Software (Vol. 25, Iss. 5), 2008

[9] Reiss, S.P. Semantics-based code search. Proc. of the Int. Conference on Software Engineering, 2009.

[10] Zaremski, A.M., Wing, J.M. Specification Matching of Software Components. ACM Transactions on Software Engineering and Methodology (Vol. 6, Iss. 4), 1997.

[11] Lemos, O., Bajracharya, S., Ossher, J., Morla, R., Masiero, P., Baldi, P., Lopes, C. CodeGenie using Test-cases to Search and Reuse Source Code. Proc. of the Int. Conference on Automated Software Engineering, 2007.

[12] Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison Wesley, 1999.

[13] Hummel, O.: Semantic Component Retrieval in Software Engineering. PhD dissertation, Univ. of Mannheim, 2009.

[14] Ye, Y.: Supporting Component-Based Software Development with Active Component Repository Systems, PhD dissertation, University of Colorado, 2001.

[15] Voorhees, E.M., Harman, D.K.: TREC: Experiment and Evaluation in Information Retrieval. MIT Press, 2005.

[16] Hummel, O., Atkinson, C.: Using the Web as Reuse Repository, Proc. of the Int. Conf. on Software Reuse, 2006.

[17] Janjic, W., Hummel, O., Atkinson, C.: More Archetypal Usage Scenarios for Software Search Engines. Proc. of the International Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation, 2010.