Ricardo Eíto Brun
Universidad Carlos III de Madrid (Spain)

Retrieval effectiveness in software repositories: from faceted classifications to software visualization techniques

Abstract: The internal organization of large software projects requires an extraordinary effort in the development and maintenance of repositories made up of software artifacts (business components, data models, functional and technical documentation, etc.). During the software development process, different artifacts are created to help users in the transfer of knowledge and enable communication between workers and teams. The storage, maintenance and publication of these artifacts in knowledge bases – usually referred to as “software repositories” are a useful tool for future software development projects, as they contain the collective, learned experience of the teams and provide the basis to estimate and reuse the work completed in the past.

Different techniques similar to those used by the library community have been used in the past to organize these software repositories and help users in the difficult task of identifying and retrieving artifacts (software and documentation). These techniques include software classification – with a special emphasis on faceted classifications, keyword-based retrieval and formal method techniques. The paper discusses the different knowledge organization techniques applied in these repositories to identify and retrieve software artifacts and ensure the reusability of software components and documentation at the different phases of the development process across different projects. An enumeration of the main approaches documented in specialized bibliography is provided.

1. Software repositories and knowledge organization techniques

Software reuse is one of the key practices in the design and implementation of software systems and applications. This practice gives us the key for reusing the knowledge and the artifacts built in previous projects when designing new solutions.

To move from the textual representation of the software requirements to the final code, different artifacts must be created. These artifacts represent the knowledge embedded in the functional specifications, and are intermediate steps to reach the final representation in programming code.

Standard artifacts not only include the programming code, but also the functional and technical documentation, detailed specifications, test cases, etc. Software reuse techniques can be applied to the different artifacts created during the development process: from functional specifications to test cases, and not only to the source code itself. Regardless the scope of the software reuse initiative, it requires the set up and maintenance of a software repository or library where the different artifacts are stored and managed; this repository must provide an efficient mechanism to enable the identification and retrieval of the stored artifacts.

In this way, a software repository can be seen as an information system where users can access the software components and their related information using a retrieval subsystem. This makes these repositories similar to the information retrieval components traditionally used in document and bibliographic management systems. In the following sections we provide an overview of the main retrieval mechanism proposed during the last years for the design and implementation of the retrieval subsystem in software repositories.
2. Keywords and controlled vocabularies

One of the first approaches to solve the problem of information retrieval in software repositories was based on the use of keywords manually assigned by trained users to the different items stored in the repository. The use of controlled vocabularies to restrict the set of terms to be used as accepted descriptors was part of these early initiatives. Once the items were indexed, the users of the software repository should use the same keywords or indexing terms to formulate their queries. The GURU system is one of the best known examples of this keyword-based approach. Other proposals – like LASSIE (Large Software System Information Environment) and NLH/E – implemented more complex solutions with the adoption of list of synonyms and thesauri containing term relationships.

3. Use of natural language

The use of full-text indexing techniques was the second step in the development of retrieval subsystems. People behind this approach remarked that it was difficult for end-users to use terms extracted from a controlled vocabulary; they also noted the subjectivity behind the indexing process based on manually assigned descriptors.

Today, automatically indexing the full text of documents and information related to software components is one of the standard approaches to face the challenge of information retrieval in repositories. Systems based on this approach usually offer the capability of processing the file with the source code and extract the comments entered by the programmers; these comments will be indexed to obtain relevant keywords. To identify these comments and metadata, they must be delimited or marked with special characters. In the case of documents, the indexing process can be executed against the full text.

The main problem of this approach is that the comments and documentation related to a software component usually does not have the appropriate size to get all the benefits from the full-text indexing techniques, what makes difficult to identify relevant terms based on their frequency. To face with this issue, Singleton (Singleton, 1993) proposed a solution based on the retrieval of keywords obtained from the names of the software components. In his proposal, the keywords gathered from the components’ name built the main inverted index. To solve the problems of synonym and homonym terms, the system also incorporated a dictionary with equivalences, term specializations, abbreviations, etc. These dictionaries provided users with a tool to extend the search by adding to the query terms related to those initially entered by the user.

People who think that full text indexing is a better choice than controlled vocabularies have indicate the need of keeping repository maintenance costs as low as possible, as well as the possibility of obtaining relationships between terms automatically by means of the co-occurrence measures (Henninger, 1994). The indexing system would be in charge of building this knowledge structure to supports a flexible searching process (adding an additional value to those retrieval systems based on the use of Boolean operators and ranking algorithms).

Today, full text searching and indexing is a technology used in most of the companies and organizations, and this technology has become a commodity adopted by most of the software reuse projects. One of the most interesting projects where these techniques were used is the Agora Project from SEI (Software Engineering Institute, Carnegie Mellon University); in this project, full-text indexing based on Altavista technology was combined with the dynamic access to the Java-based components’ interfaces (Seacord, 1998).
4. Faceted classifications

The best example of the impact of classical information retrieval techniques in the creation and maintenance of software repositories is the use of faceted classifications to organize the items in the repository. The author who led this approach was Prieto-Díaz (Prieto-Díaz, 1987) at the end of the eighties. In this approach, each software component is classified by means of a “notation” that contains information about the class to which the artifact belongs. This notation is built by assigning terms or keywords to the different facets or “aspects” used to describe the software component.

The model proposed by Prieto-Díaz was initially based on the use of six facets. Each faced was linked to a set of terms or predefined values (controlled vocabulary). In addition, for each accepted term there was a list of synonyms that helped users in charge of classifying the artifact choose the appropriate terms for each facet. These synonyms were also available for the end-users of the retrieval subsystem when exploring the contents of the repository.

This model also included information about the similarity (based on the co-occurrence of words) between the terms accepted for each facet. The purpose of managing this measure of similarity measure was to automatically expand queries in those cases in which the items in the repository did not include the terms used in the query. The query could be expanded automatically by adding those terms closer to those entered by the user.

The system proposed by Prieto-Díaz also offered a method to sort (rank) the results; retrieved items were sorted based on how easy it was to use them. This ranking was calculated based on a set of variables assigned to the source code, among them: size in LOCs (lines of code), number of conditional sentences or the experience of the user running the query.

Authors who think that faceted classifications are a good approach to solve information retrieval issues in software repositories indicate that full-text indexing cannot be considered a definitive solution, as the textual descriptions provided with the source code are usually to short to be significant for retrieval. The benefit behind faceted classifications is that they provide the precision that is needed in software repositories – this precision requirement is greater than in standard document and bibliographic management systems -.

Past bibliography offers detailed information about projects where the faceted classification approach proposed by Prieto-Díaz was applied with success: GTE Data Services, Contel and IBM RSL.

More recent contributions also make use of retrieval subsystem based on faceted analysis. For example, Zang (Zang, 2000) describes a retrieval subsystem integrated with the MetaEdit+ CASE tool. In this proposal, the author proposes a faceted classification to describe the different objects in the repository. In addition, these items are organized in a hierarchy with three levels (component-unit level, diagram level and project level). Each item is described by means of a record made up of different facets – available facets depend on its level in the hierarchy -.

5. The need of classification and domain analysis techniques

Literature on software repositories and reuse usually put together faceted classification and domain analysis techniques. Domain analysis can be defined as a process that is applied to identify, capture and organize the information used in the development of a software system. The purpose of domain analysis is to make all this information reusable in the design of new systems.
Domains can be defined as *areas for which a software system or application is designed; the domain must have clearly established limits.*

Domain analysis techniques were introduced by Neighbors in 1981 (Prieto-Díaz 1990); this author defined this technique as “*an activity that consists in the identification of the objects and operations of one class of similar systems, in a similar problem domain*”. Domain analysis had as an objective the reuse of the artifacts created during the analysis and design phases, and not only the reuse of the programming code. Prieto-Díaz gave a wider scope to this definition by adding to its objectives the “*development of an information infrastructure that allows the reuse*”; the result of domain analysis would include “*domain models, development standards and repositories of reusable components*”. In another document, Prieto-Díaz described this technique as “*the selection, abstraction and classification of functions, objects and relationships in a similar way to that used by librarians to design specialized classification systems*” (Prieto-Díaz 1994).

Today domain analysis refers to one activity completed during the analysis phase – at the beginning of the software development process – to identify the main classes or entities that the software system must handle. This concept is not necessarily related to the reuse of software artifacts or to the need of organizing the result of these analysis tasks to enable reuse.

Domain analysis must solve the problems usually found in the set up of a software repository. One of these issues is the representation of the knowledge embedded in the different artifacts that are created during the project. When domain analysis was proposed, authors recommended the use of the methodologies applied at that moment: SADT, structured analysis, etc. Regarding the classification and retrieval capabilities of the system, the main approaches were based on faceted classifications.

One of the best example of models of repositories based on domain analysis was the DARE system (Frakes, 1998); DARE was not only a tool but also a methodology to complete domain analysis. It included a subsystem to extract terms from the programming code and related documentation, and the automatic classification and aggregation of terms by means of clustering techniques.

### 6. Formal Methods

In the middle of the nineties, the use of faceted classifications became the “standard” method to organize software repositories and enable the efficient retrieval of artifacts. The main alternative to this method was based on the use of formal methods. Formal methods are based on mathematical representations of the systems (Wing, 1990). This technique gives us the choice of detailing the initial specifications, the design and the test cases (for later verification and validation) of any system. Formal method representations should be used when indexing components and also to run the queries against the repositories.

The use of these techniques in the organization of software repositories has numerous supporters and documented examples.

The advantage of formal methods is greater precision in the representation of artifacts (compared to specifications based on natural language). Its main drawback is that building artifact descriptions and queries with formal methods is difficult for both cataloguers and end-users of the retrieval subsystem.

Retrieval subsystems based on formal languages are divided into two groups (Hemer, 2001):

a) Systems where descriptions of components are based on their signatures – that is to say, on the input and output parameters or arguments used by the component –
and

b) Systems where the description of components are based on the behavior of the components. This type of subsystems offers more detailed descriptions as they include pre- and post-conditions that specify the initial and final status of the components.

In both cases, the retrieval process starts with the formulation of a query also expressed in a formal language; the query would include a description of the components that the user wants to retrieve. The system will retrieve from the repository those components whose descriptions match the query.

7. Evaluation and comparison of approaches

The different techniques applied in the organization and retrieval of information from software repositories were tested by William B. Frakes and Thomas P Pool in 1994. The evaluation process offered some similarities with the Cranfield project – and other similar initiatives – developed in the document retrieval area. The authors compared the behavior of retrieval subsystems based on faceted classifications, hierarchical classifications, metadata (property-value pairs) and keywords. The traditional criteria of recall and relevance were used to measure the effectiveness of the different approaches.

The conclusions of this study did not identify significant differences in retrieval effectiveness between the different techniques / subsystems used; this conclusion was similar to the obtained in tests completed against document and textual databases.

One interesting conclusion of this classical study was the recommendation that systems should provide different, complementary retrieval methods and techniques. This conclusion is similar to the principles expressed by Henninger (Henninger 1994 1996). He proposed a system in which the retrieval process is based on subsequent iterations of the user with the retrieval system using the traditional relevance feedback concept.

In Henninger’s proposal the system shows a set of candidate terms to the user; then, the user can select these terms to redefine the query. The prototype designed by Henninger – CodeFinder – calculated the candidate terms through an activation process based on the co-occurrence of terms in the free-text description of the components.

8. Areas of improvement: Visualization and comprehension

One of the objectives that must be reached to obtain an effective retrieval subsystem is keeping the link between the different artefacts managed in the development process. Quality of the documentation and the maintenance of the relationship between textual descriptions and programming code and components is one of the activities that can help improve the effectiveness of the retrieval subsystem.

The lack of links between documents and components results in the need of investing a lot of time reading overwhelming documents that in most of the cases do not provide a clear understanding on how a specific functionality has been implemented, as these documents are not related to each other or to the real implementation.

The use of diagrams based on standard modelling language offer the capability to understand the code and artefacts retrieved from the repository. These models contain different aggregation levels that can also be used to expand the search based in free-text comments (the item in the repository not only is described by the free-text comment directly attached to it, but also by the free-text comments and documents attached to artifacts in upper levels.
9. Conclusions
Building software repositories is one of the areas in which different retrieval, classification and knowledge organization techniques have been applied. The conclusions can be summarized in the need of having flexible information retrieval systems that give the users the possibility of reformulating queries interactively, where different retrieval techniques must be used together and combined to obtain the best results.

Another interesting point is the need of the integration between the CASE tools and the retrieval subsystems used in the repositories. This integration is interested not only from the usability point of view, but also to exploit in the retrieval process the network of related artifacts that CASE tool manage.

10. Bibliography
HEMER, David ; LINDSAY, P. Specification-based Retrieval Strategies for Module Reuse.