Towards an Effective Integrated Reuse Environment

Jorge Claudio Cordeiro Pires Mascena¹,² Silvio Romero de Lemos Meira¹,²
¹C.E.S.A.R – Recife Center for Advanced Studies and Systems
{jorge,silvio}@cesar.org.br
²Federal University of Pernambuco – Informatics Center
{esa2,vcg}@cin.ufpe.br

Eduardo Santana de Almeida¹,² Vinicius Cardoso Garcia¹,²

Abstract

A fundamental premise for any type of reuse is the knowledge about the existence of the object to be reused. Such knowledge may already be available, for example, due to the past experience of the subject of the reuse action or may be obtained through knowledge dissemination. Information retrieval is a key mechanism for allowing a uniform dissemination of the knowledge about available reusable objects.

The instantiation of this problem to the software reuse field is the subject of this work. The synergy among the software reuse and information retrieval fields is exploited in the formulation, implementation and evaluation of an integrated environment that aims at promoting a greater reuse activity level on the quest for developing software with better quality while consuming fewer resources.


Keywords software reuse, information retrieval, reuse metrics

1. Introduction

Software reuse, in all its variances, is generally regarded as the most important mechanism for performing software development more efficiently. This belief has been systematically enforced by empirical studies that have, over the years, successfully demonstrated the effects of reuse on development aspects such as quality and cost of software systems [3] [7] [10] [18] [20] [26].

Although these studies, in most cases, are restricted to small, domain-specific systems, there is a general notion that such results could somehow be extrapolated to more generic contexts. Unfortunately, as reports from large software development organizations [16] [17] [19] and empirical and informal researches [5] [14] indicate, in fact this notion is still far from the truth.

Many reasons have been enumerated [23] to justify the inability of the software development community to fully accomplish a rather simple and universal concept, at least in theory. The practice, however, has proved this to be a rather complex task, which poses a number of challenges to individual developers, development teams and organizations. Accommodating the different interests of the individuals and groups involved, at different levels, in a typical software development scenario is one of the most challenging tasks [8].

Yet, most existing solutions on the software reuse field treat reuse as an isolated concept in the development process, therefore failing to provide an effective solution by addressing the reuse problem from an integrated standpoint. From the list of requirements for an effective reuse program [1], two complementary aspects are particularly important and deserve special attention: (1) reusable software location and (2) reuse activity assessment.

The lack of knowledge about available reusable software that is suitable for a particular task combined with the notion of the cost for achieving reuse are the causes for the number one software reuse failure mode [9]: no attempt to reuse. Developers face the problem of locating, understanding and incorporated reusable software into newly created software. The available knowledge is usually unstructured and spread in information islands. For this reason, the problem of locating reusable software is a key aspect for an effective software reuse program and integrated reuse environments may be employed in order to close this gap.

This article describes the proposed Asset Development Metrics-based Integrated Reuse Environment (ADMIRE), from the specification of its requirements to the experiments conducted in order to evaluate its performance.

2. ADMIRE Requirements

The main requirements of the proposed ADMIRE environment are based on the authors’ experience on software development and reuse, and on some works in the literature that present a set of requirements for effective reuse environments, search engines and reuse repositories [13] [21].

2.1 Extensibility

Software development involves a large number of steps and different intermediate types of asset are produced along the path. There are usually multiple alternative formats to build each type of asset and on top of that, the information contained in the assets may be encoded in different spoken languages.
Examples of asset types include use case specifications, documentation and programming units. A programming unit, in its turn, may be defined in a COBOL program or a Java class, for instance and the method names in that Java class can be written in English or Portuguese, for example.

For this reason, the proposed solution must be generic enough to accommodate different asset types, formats and languages. Hooks must be provided in order to allow the addition of new capabilities to the environment to deal with them.

2.2 Precision
Information retrieval performance is measured in terms of the achieved precision and recall and the solution heavily depends on that to (1) assess the reuse potential of a set of assets given a legacy asset repository and (2) help keeping a high level of reuse activity by actively delivering reuse candidates for the user tasks.

Since large organizations tend to have a huge amount of legacy assets available, a poorly designed solution would lead to a vast amount of irrelevant information being delivered to users.

2.3 Ubiquity
No attempt to reuse is the main software reuse failure mode [9] and that is mainly caused by the lack of knowledge of existing assets to be reused. A reuse within development approach [27] must be employed, minimizing the cost of finding reusable assets.

The proposed solution should then provide programmers with a comprehensive set of tools that smoothly merge with existing development environments and tools in order to minimize the effort for achieving higher levels of reuse activity.

2.4 Dynamicity
The cost for developing reusable assets usually prevents organizations from adopting a systematic reuse program. Moreover, projects teams incentives diverge from the creation of reusable assets [8]. Their main motivation is delivering the project’s specific products on time and the overhead for making such products reusable is usually neither affordable nor desirable.

To overcome this, the solution must take an incremental approach into making assets more reusable as new opportunities for using them are presented. It must be able to leverage the assets that have not been specifically designed or developed for reuse, promoting a greater reuse activity in organizations.

2.5 Continuous Metrics Extraction
From the organizational perspective, the reuse activity must be systematically monitored so the impacts of reuse over other development aspects, such as quality and cost, can be assessed and deviations can be timely detected and handled.

For this reason, automated metrics extraction tools must be provided by the solution in consonance with existing continuous integration practices [6].

2.6 Platform Independency
Organizations usually have heterogeneous platforms, so the more platforms are covered by a reuse environment, the more developers can benefit from it.

The implementation of the environment functionalities must then be based on technologies that are easily portable across existing platforms.

2.7 Summary of Requirements
From the requirements listed in this section, it is evident that ADMIRE’s ultimate goal is to provide users with a comprehensive set of functionalities that focus on facilitating the reuse activity from the individual perspective and the reuse management from the organizational perspective. Next section discusses a proposed reuse metric to support the environment.

3. Proposed Metric
Existing reuse metrics [22] have in common the fact that they basically aim at assessing, although in different ways, how much was reused during the construction of a software product (usually an application). For this reason, they can all be seen as realized (or achieved) reuse metrics.

While this is a fundamental aspect to be considered when assessing the reuse activity, there is a critical detail that is lacking on all proposed reuse metrics so far: the reuse potential of a product relative to a repository of legacy assets. That is, given a repository of (semi) reusable assets, the following question must be answered: “How much could be reused when building this new application?”.

Once the need for defining a reuse potential (rp) metric is agreed upon, the remaining issue is how to perform the calculation of such metric and for that, some information retrieval concepts must be employed.

The artifacts produced in a specific project belong to the query space, while the assets available in a repository belong to the search space. The proposed metric is defined in terms of the set of queries extracted from the query space and, from this set, the number of successful queries against the search space.

The proposed metric builds on the concepts exposed so far and depends on some additional definitions:

- Given a collection of artifacts (cart) (e.g. a Java program), the query set (qs) of that collection is composed of the minimal set of queries that are directly derivable from that collection and that represents the contents of that collection. The query set is obtained by applying a query formulation function to the cart (form(cart)).

- Given a reuse repository (rr) and a query (q), the result set of the execution of q on rr (es(q, rr)) is composed of potential reuse candidates (cand) that match q and are indexed by rr. Each cand has its associated relevance or ranking (rank(cand)) for the context of q; and

- Given a rr and a query set (qs), the successful query set (sqs) is composed of the queries in qs, such that the evaluation function on their result set (eval(rs(q, rr))) is successful.

The rp of cart relative to rr (rp(cart, rr)) is then defined as the ratio between sqs and qs.

The rp may be seen as a measure of similarity between a cart and the artifacts indexed by a rr. Reuse opportunities are greater when the similarity measure yields greater values. Figure 1 illustrates its extraction process given cart and rr. The resulting rp value is 0.5.
This metric plays an important role in the proposed environment by serving as a consolidation of the individual perspectives of a development team. Its main focus is to grasp the reuse activity from the organizational perspective in conjunction with other reuse metrics, detailed in [22].

It is important to stress that the proposed metric does not have a direct relation with other software development aspects such as quality and cost, since it only assesses the amount of reuse that can be achieved without concerning about the actual achieved amount of reuse.

4. ADMIRE Architecture

The overall goal of the architecture is to satisfy the environment requirements in a consistent way, describing how its internal components are combined. Although formal methods for deriving system architectures from requirements do exist [4], such derivation processes are out of the scope of this work.

The main phases that should be supported are: (1) legacy content retrieval, (2) repository indexing, (3) activity monitoring, (4) query formulation and execution, (5) results evaluation and presentation, (6) indexed contents retrieval, and (7) metrics extraction. Figure 2 depicts the environment architecture. The dashed arrows represent operations that are not implemented in the initial version (Retrieve, Write, Store, Get, Index and Read). The Crawler and the Tracer modules, faded in the figure, are also not available in the initial implementation.

Each developer has a limited view of the Working Set (DWS). This view corresponds to the artifacts related to his tasks. The manager, on his turn, is responsible for monitoring the development activities and ensuring that the reuse opportunities in the Working Set are realized as much as possible.

4.1 Legacy Contents Retrieval

The legacy contents retrieval phase consists on continuously monitoring the produced assets and passing them for indexing. An asset evaluation policy is necessary for determining whether a specific set of assets is proper for future indexing.

This policy is responsible for filtering low quality assets that would negatively impact retrieval performance or cause problems if reused. It is also responsible for determining when the index should be updated. This accounts for the dynamic nature of the indexed artifacts, as stated in the dynamicity requirement.

4.2 Repository Indexing

Once the assets are retrieved, the indexing phase takes place. During this phase, performed by the Indexer module, the artifacts’ contents are parsed and analyzed before being actually indexed.

The contents of the available artifacts are converted to a common representation. This common representation is then interpreted during analysis and indexed, if considered relevant.

4.3 Activity Monitoring

The ubiquity requirement is satisfied mainly by the Listener module, which monitors and interprets user activities, like adding a method to a Java source file.

From this interpretation, queries are formulated and executed against the repository and reuse candidates can be suggested to the user. This consists on the active information delivery mechanism of the integrated reuse environment.

4.4 Query Formulation and Execution

The queries are formulated by the query formulation agent, contained in the Searcher module. The formulation is based on the contents of the artifacts being edited by the developer in a similar approach to the repository indexing phase, although the actual analysis performed may differ due to the distinct nature of development for and with reuse.

4.5 Results Evaluation and Presentation

The result evaluator agent, contained in the Searcher module, is responsible for detecting candidates that should not be presented to the user, based on the feedback provided from previous interactions or on information that is not available in the index and therefore cannot be taken into account during search time.

The remaining results from the analysis are then manipulated and finally presented to the user by the Presenter module. This module is responsible for determining how and when these candidates are presented to the user. Cognitive issues like the level of intrusiveness of the delivery must be taken into account when making these decisions [27].
4.6 Indexed Contents Retrieval
The search results are presented to the user by the Presenter module. The next step is the indexed contents retrieval phase, performed by the Searcher module, initiated upon user request. This phase is responsible for providing the system with user feedback (which assets were considered relevant to the context) and retrieving the actual asset from the repository system.

4.7 Metrics Extraction
Complementarily to the active information delivery cycle consisted of the previous phases, the metrics extraction phase takes a more general look at the produced artifacts from the organizational perspective. All previous phases, except the legacy contents retrieval and repository indexing phases, are focused on reuse from the individual perspective, aiming at helping developers in achieving a higher reuse activity.

This phase, performed by the Extractor module, is responsible for ensuring that given a cart being produced and the available $rr$, the development team extracted the most out of the repository when building the new cart. That is, good reuse candidates presented by the system have not been neglected.

5. Implementation Outline
As stated earlier, the initial implementation of the solution comprehends a subset of the features contemplated by the architecture. The main goal of the initial implementation is to provide a strong foundation for future improvements.

5.1 Common Elements
The system architecture relies on a set of common elements, shared by the core components for consistency between the extracted metrics and the information delivery mechanism.

The artifact parsing is managed by the parsing manager component that can be configured to associate file types to parsers. The initial association, restricted to Java source files, is performed by a parser generated by the ANTLR\(^1\) parser generator.

The analyzer extracts the relevant information out of the parsing result for later indexing or searching. As the parsing manager, the analysis manager can associate asset types to analyzers.

The analyzer extracts the methods’ names of a Java class. Only relevant methods, determined by the size their bodies in this implementation, are extracted to avoid simple getters and setters.

5.2 Core Components
The basic functionality of the solution is provided by a set of core components. These components are (1) Searcher, (2) Indexer and (3) Extractor. They share the elements described in Section 5.1.

The indexer and the searcher components use the Lucene text-based search class library as the core information retrieval mechanism because of its model extensibility and its vastly available documentation [15]. Furthermore, the Lucene class library is used other RiSE\(^2\) projects, such as Maracatu [12].

The searcher component is responsible for formulating and executing queries. The actual query formulation is delegated to the query formulator element, which in turn uses the same set of artifact parsers and analyzers that are used during indexing time.

The extractor component works as a batch searcher component executor. All formulated queries for a given cart are executed and the results are analyzed by the search result analyzer element, which determines if they were successful. The $rp$ metric is then computed from these sets.

The success of a query is determined by the ranking value of the most relevant document in the result set that is compared to a configurable threshold $(thr)$ value. The rationale for only considering the most relevant document is that this tends to yield higher precision results, since documents with smaller ranking values are less likely to be relevant to the context of the query.

5.3 Integrations
The environment must integrate with development and general usage tools in order to maximize the reuse activity throughout the organization. The initial integrations include a build tool, an IDE and an instant messenger. The goal is to demonstrate how the solution fits into tools and environments that are used on a daily basis by developers and managers.

For the build tool integration, an Ant task is provided to invoke the Extractor component. This task can be incorporated to the set of tasks to be performed during build time in a project. The $rp$ metric can then be computed every time the system under development is built and presented to the user.

For the IDE integration, an Eclipse plug-in is provided to invoke the Searcher component and present the results in the editor window. The plug-in listens to code changes performed by the developer and determines when the Searcher component should be invoked. An internal thread consumes a change event queue in order to decouple the query processing from the Eclipse’s event dispatching thread.

Once the query result set is available, the plug-in creates Eclipse markers that are displayed as small icons on the line of the asset (method, class etc.) that originated the query and a display text with the results from that query. Since the plug-in builds on top of Eclipse’s extensible framework, all features provided by the platform, such as the message tab that displays all active messages in a consolidate view and the hierarchical element marking, that automatically marks all ascendants of a marked file are available.

A MSN Messenger robot is provided to demonstrate a non-conventional use of the integrated reuse environment. It was built on top of the JMML\(^3\) open-source Java library.

The robot is responsible for signing in the MSN network, listening for instant messages that are treated as regular queries from the users, who must have the robot in their contact lists, and replying to these messages with the query results.

---

1 http://www.antlr.org/
2 http://www.rise.com.br
3 http://sourceforge.net/projects/pmlibs
6. Evaluation

The source code of completed projects, from the code repository of a mid-sized software development organization in Brazil\(^4\), was used for the experiments. The downside of this approach is that the impacts of the information delivery mechanism cannot be assessed, as performed in [27], because no intervention from developers occurs during evaluation.

6.1 Methodology

The methodology adopted, although innovative due to the nature of the proposed environment, is strongly inspired by existing methodologies for information retrieval [2] and reuse metrics [3].

6.1.1 The Goal

The goal of the evaluation is answer the questions: “Does the extracted \(rp\) reflect the actual similarity between projects?”; “What is the behavior of \(rp\) in relation to varying aspects such as the size of the index and the target set of artifacts?”; and “How effective is the underlying information retrieval mechanism?”.

The \(rp\) extraction process must be evaluated in different configurations, using the selected projects as both the \(rr\) and the \(cart\). The parsing and analysis functions used for indexing and query formulation must be evaluated as well. This part of the experiment can be thought of as the coarse-grained evaluation of the environment. The fine-grained evaluation will be based on traditional information retrieval evaluation [2].

6.1.2 Configuration

Metric extraction will be performed on all possible permutations with repetition of pairs of artifacts sets from the set of selected projects. For each permutation pair, one artifact set will act as the \(rr\) and the other one as the \(cart\). Furthermore, different values for the query success \(thr\) value on the metric extraction process will be used in order to estimate the stability of the metric according to such variation.

Once the \(rp\) metric is calculated for each permutation pair, a manual inspection by sampling the target artifacts, manually formulating an executing the queries and checking the result set in order to calculate the precision and recall obtained by the information retrieval mechanism will be performed.

6.1.3 Variables

Experiments usually employ a set of variables that link causes and effects [24]. Independent variables are attributes that are manipulated to create different scenarios. Dependent variables, on the flip side, are the output values expected to change according to changes to the independent variables.

In the context of this experiment, the independent variables are: \(cart\) size, \(rr\) size, both varying from project modules to complete projects, and \(thr\), varying between 0.5 and 1. The dependent variables are: \(rp\), precision and recall.

6.1.4 Results Visualization

Two techniques will be used for results visualization. The first technique, used for the evaluation of the precision and recall of the information retrieval mechanism, follows the traditional information retrieval approach, displaying a table with achieved precision and recall mean, standard deviation and variation [12].

The second technique follows the approach in [11], making use of a color scales to represent \(rp\) values with the goal of making the data comparison process a perceptive task (i.e. visual) instead of a cognitive task (i.e. numerical).

A reuse potential matrix (rpm) will be created for each configuration. Lines correspond to entities as the \(rr\) and rows correspond to entities as the \(cart\). A color temperature scheme will be used in order to make \(rp\) identification more intuitive: cold temperature colors (dark shades in grayscale) correspond to low \(rp\), while hot temperature colors (light shades in grayscale) correspond to high \(rp\).

6.2 Results

The \(rp\) between all pairs of artifact sets was collected and the \(rpm\)s were rendered. Figure 3 shows the \(rpm\) for the \(thr\) value of 0.5 at the module level.

\[\text{Figure 3. Module-level rpm (}\ thr=0.5\text{). rpm(D3, A9) is low (P4), while rpm(D2, D2) is high (P3).}\]

The precision and recall of the underlying information retrieval mechanism were calculated from the results of 20 queries randomly selected by the contents of the known projects existent on \(rr\). Table 1 presents the values. Since the mean recall is not greater than the mean precision (even considering the standard deviation and variance), it can be concluded that the precision of the information retrieval mechanism is greater than its recall. This is the expected behavior, since high precision is one of the requirements of the solution, as stated in Section 2.2.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>0.8976</td>
<td>0.2515</td>
<td>0.0632</td>
</tr>
<tr>
<td>Recall</td>
<td>0.3055</td>
<td>0.2821</td>
<td>0.0796</td>
</tr>
</tbody>
</table>

\(^4\) C.E.S.A.R (http://www.cesar.org.br), with 600+ developers
7. Related Work

CodeBroker [27] combines the information available in Javadoc comments with a signature matching mechanism to actively suggest reuse candidates. ADMIRE contemplates different types of artifacts and the internal structure of the artifacts is used for filtering during the indexing and searching phases.

RewCalc [25] is a web based reuse metrics calculator that depends on the user initiative for the assessment of economics oriented reuse metrics. ADMIRE extracts the rp metric and integrates to continuous integration tools in order to facilitate the metric extraction process in a consistent manner.

8. Conclusion

This paper proposed an integrated reuse environment - ADMIRE – as a tool for helping organizations in achieving an efficient software development practice.

ADMIRE is part of a broader context, the RiSE project [1], which aims at guiding organizations form a chaotic reuse activity to a more systematic one, employing processes and tools [12] [13]. The evaluation demonstrates that ADMIRE can be used as a basis for providing momentum at the initial stages of this process as well acting as a supporting tool for the later stages.

References


