Abstract

The development of large software systems is a typical example for collaborative development efforts. Moreover, software development becomes more and more component-oriented. The source for these components can be either a component repository inside an organization or one of the emerging open source development networks available in the Internet. In this paper we describe an approach to search for potentially useful components during a software development project. Our approach is by no means restricted to components in an implementation oriented sense, but covers all types of artifacts created within a software development process. It is based on retrieval techniques for structured documents and uses contextual information about a user and his or her current work to refine the queries.

1 Introduction

Today everyone would agree that the reuse of previously developed artifacts can significantly improve productivity during software development. An open question in this respect is how to find well-suited previously developed artifacts in a concrete situation. In our opinion, customized information retrieval facilities can be useful for this purpose. The idea is that software developers — e.g. the software developers in a global company — open their local repositories for each other. In this way a software developer searching for a class can benefit from the artifacts developed by all software developers in the community. Another opportunity would be to exploit further sources for software artifacts like open source development networks with their worldwide community.

The main problem in this scenario is to find appropriate artifacts, which are useful in the concrete situation. To this end, the structure and the relationships between the artifacts in the repositories and the working context of the software developer can be used. For example, the dependencies between requirements documents, analysis documents, design documents, code modules and test protocols will be represented in a repository. Given a software developer in charge for the design of a system, we can look for existing design documents developed for similar analysis documents and requirements. Furthermore, we can increase the precision of this similarity search enriching the similarity condition by knowledge about the current working context of our software developer.

To become more concrete, let us consider the scenario sketched in figure 1. This figure depicts the current working situation of a software architect within a software development process. The software architect is currently working on a design document which implements the system defined in an analysis document which in turn depends on two requirements documents. On the other hand, there are the software artifacts from earlier projects given on the right side of figure 1. These artifacts are part of a document network representing the dependencies between the requirements, analysis, design, and implementation documents.

In this situation we want to assist the software architect as follows: Based on the documents building the specification for the design document under work (the three upper documents on the left side of figure 1), we perform a similarity query searching for similar requirements and analysis documents in the pool of documents from finished or proceeding projects (on the right side). From the requirements and analysis documents found in this way we can determine associated design and implementation documents. As a consequence, we can provide the software architect with documents ranging from requirements documents to implementation documents which might be helpful...
with his or her current task.

Furthermore, this scenario can be augmented in two directions: First, the pool containing the artifacts of finished or proceeding software projects has to be realized as a distributed pool containing artifacts of various users sharing the results of their work. This leads to a loosely coupled human cooperation over the Web and forms a platform supporting the remote cooperation of software developers. Second, it seems to be promising to improve the similarity query in our scenario by the additional consideration of information about the working context of our software architect. For example, information about his qualification or his role in the project team might give helpful hints for a more precise similarity search.

In the rest of this paper we will address the various aspects of the system sketched above in more detail. A first important aspect concerns the relationships between the artifacts generated during software development. This aspect will be addressed in section 2. Given the structure of these artifacts the next open point are similarity queries on structured documents (cf. section 3). Thereafter in section 4 we will elaborate on the integration of context knowledge into the similarity search and in section 5 we will sketch our prototype implementation. A discussion of related approaches (section 6) and an outlook on future research aspects (section 7) conclude the paper.

2 Software Development Process

Today’s software development processes, e.g. the unified software development process [9] and extreme programming [2], are characterized by a methodology that postulates an iterative and incremental proceeding. The basic idea behind this postulate is to split large projects into smaller parts. These parts, often called mini-projects, will be treated as stand-alone processes and are subject to one or more iterations within the process. Hence, each iteration addresses either a different aspect of the system or advances an aspect incrementally.

Considering the unified software development process, each iteration is represented by one of the following four phases. At the end of each phase a milestone is set which includes the presentation of several pieces of information, also called artifacts. These artifacts can be retrieved as reusable components by our retrieval facilities.

During the inception phase a vision and the business case for the final product is presented. This phase gives also answers about the objectives of the system to be build, first thoughts about the architecture and the further project plan including costs.

Within the elaboration phase most of the use cases for the product are gathered. Furthermore, the architecture for the system will be designed, resulting in a baseline architecture.
The objective of the construction phase is to build the system according to the baseline architecture. The result will be a system that has beta status.

Through the transition phase problems are fixed and minor changes are incorporated into the product, resulting in a final release.

Each of these phases contains a number of activities that will be embraced by workflows, that are more or less involved in each of the phases. Activities within the workflows output a series of artifacts. These artifacts are again the input for other activities, either within the same or another workflow.

The requirements workflow attempts to express the requirements for the system in terms of use cases and textual descriptions. The resulting artifacts are use case diagrams describing functional requirements that can be detailed e.g. by sequence diagrams. Furthermore, there are other requirements such as non-functional requirements that will be gathered by textual descriptions as well as the architecture of the system.

In the analysis workflow the collected requirements will be analyzed, refined and structured. Results of this workflow are analysis package and collaboration diagrams, interaction diagrams, textual descriptions for special requirements and the architecture.

The design workflow concretizes the system describing how it should be built. Within this workflow different structural and behavioral diagrams, like class, deployment or sequence, statechart and activity diagrams, will be created. Additionally, there will be textual documents like the architectural description.

The implementation workflow concerns the implementation of the system and its deployment and integration in an existing software environment. Resulting artifacts are components that are written in a specific programming language. Additional textual descriptions will be created treating architectural aspects and the integration of the software components.

The test workflow addresses the testing of the software components against the required functionality. This includes planning, design, implementation and performing the test cases. Emerging artifacts are test cases and procedures as well as test components and plans.

Figure 2 illustrates an exemplary activity within the design workflow. Inputs of this activity are the requirements pointed out in use case diagrams and textual documents as well as diagrams and documents resulting from the analysis workflow. Outputs of this activity are class and deployment diagrams as well as further textual architectural descriptions. In such a manner we can understand an activity as a transformation of the input artifacts into output artifacts, creating implicit relationships between the input and output artifacts. Using the unified modeling language (UML), the relationships between them can be expressed explicitly through traces which indicate either a historical or process relationship between two elements that represent the same concept [3].

Throughout an activity such as the one stated above a software architect has to consider various reuse possibilities, concerning the reuse of subsystems or components already developed in other iterations within this or other projects. Finding applicable components or products includes not only their functional and non-functional description. Furthermore the whole path of artifacts created along the software development process leading to the components or subsystems is relevant. In this manner we can also broaden the area of reuse from software components or products to all artifacts created in a software development process. Figure 1 demonstrates the current working situation of a software architect creating a design document. In order to admit the reuse of already developed artifacts in the current design, the software architect has to be aware of resulting artifacts from software development processes that are still proceeding or finished. On the one hand, this takes place by the personal involvement of a software
engineer in the software development process. On the other hand, additional tool support might be useful.

Our approach of a supported collaborative software development uses a similarity search that compares the artifacts within prior workflow activities from the own process (the three upper documents on the left side of figure 1) to artifacts from other processes that may still proceed or have finished (the five upper documents on the right side of figure 1). The results of this similarity search will contain the artifacts from within the scope of the search and their resulting artifacts. This similarity search uses additional contextual information of the user — in our example the software architect — to automatically trigger and refine the search result.

3 Information Retrieval for Structured Documents

As mentioned in the introduction, our approach to support the reuse of artifacts is based on retrieval facilities for structured documents and on the use of context information. The latter aspect will be discussed in section 4. The retrieval facilities for structured documents are based on similarity queries [7].

The principle behind the retrieval can be explained by an example. Let us assume that we are searching for a reusable design document. In this case the analysis documents defining the functionality of the desired design document do already exist. These documents can be used as query documents and we can search for design documents in the pool with the artifacts of finished and ongoing projects which are connected to similar analysis documents. The search starts with the search for analysis documents in the pool which are similar to the query documents. This similarity search can be based on the textual similarity defined e.g. according to the vector space model [14]. This model is based on term co-occurrences in the compared documents. For the analysis documents in the pool so-called retrieval status values are calculated. These values are high for documents having many terms in common with the query documents. Based on these retrieval status values for the analysis documents retrieval status values for the related design documents can be derived. To this end, the average over the retrieval status values of the connected analysis documents can be used to calculate the retrieval status value of each design document. Based on these derived retrieval status values the design documents can be ranked and the design documents with the highest retrieval status values might be good reuse candidates.

This principle described in detail in [7] can be refined in many directions. First, special similarity measures can be used for different types of artifacts. Furthermore, different types of related artifacts can be used to derive the retrieval status values for the objects of the desired type. If we consider the example given in figure 2, the retrieval status value of design class models can be calculated based on the retrieval status values of the related use-case diagrams, requirements documents, analysis models, and architecture descriptions. Finally, additional context information can be incorporated into the calculation of the retrieval status values. This aspect will be addressed in more detail in the next section.

4 Exploiting the Context

As mentioned earlier, we use information about the context of a user to trigger and refine the search for similar artifacts. As a starting point for our context-aware retrieval engine we use a unified user model covering several dimensions from the user, his or her working context and the interaction with applications he or she is using during his or her work [6].

Figure 3 illustrates our context model with its different context dimensions. In the following we introduce the context dimensions of our user model and their appropriate concepts within a software development process.

The user context comprises the physical and orga-
nizational context of the user as well as his or her personal profile. Considering a software development process the physical context contains e.g. geographical information about a user’s working office. The organizational context represents the position a person holds within the organization as well as the roles he or she holds within the software development process. The roles a person fulfills within a process flow are an essential information about his or her responsibilities. Finally, a user profile can give information about a user’s knowledge and skills. In our model the personal profile stands for the user’s self assessment whereas the task dependent profile represents information about the knowledge and the skills derived from activities he or she handled before.

The working context characterizes the current activity a person performs. During a software development process we can assume that activities are mostly planned and well described. Furthermore, there will be tasks that are unplanned and only vague information might be available.

Finally, a person’s current interactions with the application systems supporting his or her present activities are reflected in the interaction context. Typically, these are events from menu and dialog interactions on a lower system level, e.g. creating a class diagram or editing a file of source code.

4.1 Representing the context information

To represent the context information of a user we distinguish the activities he or she performs or has performed and the artifacts that result from these activities. Both are represented as structured documents based on the extensible markup language (XML).

The dimensions within our user model that cover the user and working context as well as the interaction with applications are represented by a network of statements using the resource description framework (RDF) based on XML. An RDF statement can be seen as equivalent to a simple sentence within a naturally spoken language. It predicates about a subject in conjunction with an object. This object can either be a literal, i.e. a final value, or another subject of different statements. In this respect we can create a network of statements expressing relationships between different statements.

Another aspect of using RDF to represent context information is the use of uniform resource identifiers (URI) for subjects, predicates and objects within statements in contrast to terms from a naturally spoken language. Using terms as identifiers is still affected with problems due to the vagueness of naturally spoken languages. Using RDF Schemas the vocabulary within the user model can be unified — e.g. equivalent actions will be expressed through the identical URI rather than using different terms with an equivalent meaning. Due to the definition of an RDF Schema with a naturally spoken language it is possible for us to transform statements given in RDF to term vectors used in the vector space model [14].

Applying an ontology to the vocabulary used in RDF statements allows to bring the concepts and hence the statements within the different dimensions of the user model in a semantic relationship to each other. Furthermore, applying an ontology is an appropriate way to expand the queries with related concepts (cf. [11]).

The resulting artifacts from activities, e.g. use case or class diagrams, are inately structured documents. Also textual descriptions enclose a structure, that separates each requirement and its description within a requirements documentation. Artifacts are stored in a wide range of document formats that comprise text processing formats as well as formats for diagrams and program code. Our approach transforms the different types of artifacts into XML, providing an XML Schema for each artifact type. Thus a formulation of queries for structured documents over different types of artifacts is feasible. Each artifact in the model index is addressed with a unique identifier (URI) that can either be referenced from other artifacts or from RDF statements within the user model repository.

4.2 Obtaining context information from client applications

A modern software development process is unthinkable without the support of tools [9]. A wide range of tools support the different workflows of a software development process, starting with project management tools, tools for structured requirements gathering, UML modeling products for analysis and design as well as integrated development environments (IDE) for implementing and testing the source code. The functionality of these tools can typically be extended by plug-ins.

To obtain context information from such software products used by software engineers on their clients we propose the use of plug-ins. A plug-in that resides within the client application cannot only easily gather information about the users working context but also record the interaction between the user and the application with the objective to infer actions or procedures a user performs.

For example, using an integrated software development environment that supports UML modeling as well as implementing source code and testing, we can use a plug-in to infer the following context information:
• **User and Working Context:** Using the project metadata within the IDE we can retrieve the project name, its project repository and the current user identification, which can be used to augment the working context with information about the user and his or her context.

• **Working Context:** Reviewing the created UML diagrams and their relationships stated by traces between different model elements we can infer the software development workflow the user is currently working in.

• **User Interaction:** Retrieving the open windows within the IDE and the interaction of the user with the different windows, e.g. drawing a class diagram or editing source code, we can infer what a user is currently working on or what function he or she is currently using within the IDE.

Beyond this a plug-in can access the data that is stored within diagrams, their elements and source code. This includes the documentation behind a use case as well as the documentation within source code.

### 4.3 Obtaining context information from server applications

Besides the client application, valuable sources for obtaining information about the working context are data stored within server applications. This includes the repositories in which the model data for requirements, analysis and design as well as the source code of the implementation are stored. Within such repositories metadata including the date, the creator, and the contributors of each artifact is recorded. Retrieving the content from such repositories it is possible to reconstruct the responsibilities of people to their artifacts in finished or proceeding projects.

Using the data stored in an organizational directory, e.g. through LDAP, we can additionally gain information for the user context about the positions and roles a person holds within an organization. Combining this information with data from a groupware or project management server conducting a software development process, we can further gain detailed information about the activities a person was and is still involved in.

In our approach, we use this information in two ways. As stated above, the data within the repositories will be transformed into XML and stored in an index repository. The metadata from repositories, or a project management server will be transformed into RDF statements and incorporated into the working context dimension of the user model.

In the prototype described in the next section the context information is used to restrict the query artifacts and to refine user queries. For the latter aspect query terms typical for the current user context are given more weight in the queries refined by means of the user context.

### 5 Prototype Implementation

Figure 4 illustrates the prototype for our approach to support collaborative software development by context-aware information retrieval. This prototype is still under construction but has already implemented the fundamental points of our approach.

As indicated in section 4.2, we use an integrated development environment that supports the requirements, design and analysis workflows with UML modeling as well as the implementation and testing workflows (cf. [4]). This IDE allows the integration of plugs. Our plug-in monitors the following data and interaction between the user and the IDE:

- The *project* a person is currently working on. From the metadata associated with this project the repository for model and code data as well as the user identification are extracted.
- The *type of the activities* a person is currently performing. This includes the opened windows and diagram types allowing to conclude the type of workflow and activities a person is currently involved in.
- The *interaction* between a person and the IDE comprising, e.g., the creation of new elements in a diagram or source code. This information about an interaction is augmented by the data embedded in the elements a user is working on. For example, when a user is editing a class diagram the class documentation and the relationships between the classes are included.

To refine the assumptions about the users current activity a rule based system is used. For example, opened windows within the IDE containing analysis collaboration diagrams and the simultaneous work of the user on a class diagram lead to the assumption that he or she works within an activity of the design workflow.

This contextual information is formulated as a series of RDF statements expressing the current working and interaction context. These statements are transmitted by the plug-in to the index and search server that incorporates them into the user model repository.
Within the index and search server there are two main repositories, holding the user model and the indexed artifacts retrieved from external repositories. These external repositories are connected by an indexer and connector component. The indexer component has the responsibility for retrieving and transforming data into XML according to a given schema or into RDF statements. An underlying connector supplies a common access mechanism for the indexer to the data stored within the different external repositories. For example, retrieving data from a CVS needs a different access strategy than querying an organizational directory through LDAP.

Within the query engine we use the approach for querying structured documents presented in section 3. In our approach, the current working context is used to construct a query for the index repository finding similar artifacts resulting from similar working contexts according to the working context of the user and artifacts created by the user. In contrast to permanent queries for related artifacts the query engine starts only a query procedure when the working context has significantly changed.

Query results including short textual summarizations of an artifact are sent back to the plug-in within the IDE that displays them within a message pane at the bottom of the IDE main window. By selecting such a result the user can either retrieve more information about the artifact — e.g. the information about the creator or the original storage location — or can display a preview. Previews can be generated by an appropriate XSL transformation according to the XML schema used to store the XML data within the index repository. In this way either a preview of UML diagrams can be displayed in a web browser through a transformation into the Scalable Vector Graphics (SVG) or Java documentation (JavaDoc) through XHTML.

6 Related Work

Our approach for supporting software developers with their work encompasses ideas from two main research areas.

First, there are different approaches to use information from the current working context of a user to gain information that might be useful.

The Remembrance Agent [13] represents a category of systems that use the text of a document or passages from the document a user is currently editing or reading as the context for the continuous retrieval of related documents from locally maintained document and email archives. Another characteristic for this type of context-aware retrieval systems is the non-intrusive presentation of the resulting list to the user, typically using a separate window pane.

The Lumi`ère Project [8] provides an assistant for users working with an office software suite. The use of a Bayesian user model allows Lumi`ère to derive a user’s need from the actions a user performed within an application and to infer suggestions or help texts. Abecker et al. [1] as well as Maus [12] present the use
of workflow context information to provide users with context-sensitive relevant information.

The second research area comprises the search for relevant components in repositories containing software components. The proposed approaches and systems differ in the utilized features extracted from the components. Text- and structure-based systems can be distinguished. Text-based approaches use either the description included within the component [15] or from an external documentation [10]. Structure-based systems, like [5], create a knowledge representation from the components within a repository.

Finally, a system named CodeBroker combining both research areas has been presented by Fischer and Ye [15]. This approach extracts documentation texts and method signatures from the source code a programmer edits. Therewith, similar components from a component repository are searched. The results are displayed within a portion of the editor window allowing the programmer to either refine the query or use the component. An additional user model allows each programmer to adapt the retrieval mechanism.

In contrast to these approaches and systems our approach differs in the following aspects: (1) The consideration of all artifacts created within a software development process. (2) The use of a wide range of user activities and interactions from an IDE. (3) The integration of different repositories and server applications to combine model and code data as well as information about the historical and current context of the user. (4) A user model using different contextual dimensions that will directly influence the retrieval of similar artifacts.

7 Future Work

In the present paper we have described our approach for a context-aware information retrieval system supporting collaborative software development. At present the integration of context information is based on retrieval techniques for structured text. In the future we will extend the approach to the complete range of UML diagrams. Furthermore, sophisticated techniques for the transformation of RDF-statements into similarity queries will be addressed and recall/precision tests will be performed to assess the benefits of a context-aware retrieval system.

References


