Modeling Context in Software Reuse

Eduardo Cruz[1], Vaninha Vieira[1], Eduardo S. de Almeida[1],
Sílvio R. L. Meira[1], Ana Carolina Salgado[1], Patrick Brézillon[2]

Informatics Center – Federal University of Pernambuco (UFPE) - Brazil
Laboratoire d'Informatique de Paris 6 – Université Paris 6 (LIP6) - France
{ecrs,vvs,esa2,srlm,acs}cin.ufpe.br,
{patrick.brezillon}@lip6.fr

Abstract. Software reuse research is focused in building new software based on software artifacts previously made, in order to achieve software quality and productivity. In this field, software reuse repositories are used to store reusable software artifacts for later search and retrieval. Unfortunately, it is not common to use context neither to enrich the semantics of the software artifacts stored in the repository nor to improve the possibilities of assembly between assets for further retrieval. Assembling can be seen as a processing of contextual knowledge, which can be composed by contextual elements, that is need for a given focus. In this work we propose the improvement of an existing software reuse repository inserting context information in it.

1 Introduction

Software reuse is not a new issue in software engineering, in 1968 Douglas McIlroy discussed in your work intituled "Mass Produced Software Components" [1] saying that "*the software industry is weakly founded and one aspect of this weakness is the absence of a software component sub-industry*". By that time the term software component was very related to source code and executable software. And now the term software asset has been adopted to express the idea that any artifact of the software development life cycle like: use case diagrams, test cases or software requirements documentation can be reused.

Since there, software reuse research has been trying to achieve high productivity and quality, focusing on areas like: component based development, component certification, software product lines, component search and retrieval and asset repositories. In 2004 Almeida et. al proposed the RiSE[1] Framework [2] to achieve a systematic software reuse adoption. This framework is composed of non-technical aspects like adoption process in an enterprise and technical ones like component certification process, asset repository system, tools to help the user in a software reuse environment and best practices.

Here we focus in asset repositories as the basis of the framework where our future research will evolve. Asset repositories are intended to store information

[1] http://www.rise.com.br

and artifacts of the software development life cycle that can be used during application development. In order to retrieve the information of these repositories the system needs to receive the users queries and match it with the stored artifacts to find some information that should be the expected result. However, not every information that is stored is relevant for the search. This same concept can be seen out of the software engineer frontiers. In artificial intelligence, Brézillon and Pomerol [3] propose a definition of context and divided it in *external knowledge*(EK) and *contextual knowledge*(CK). CK is the part of the context that is relevant for the current focus while EK is the knowledge that is not relevant for the current focus. In a common search, the system may use the content of the artifact to compare it with similar artifacts. In contextual search, we may use contextual knowledge to assemble artifacts that are only related by its context because CK acts as a filter that defines, at a given time, what knowledge pieces must be taken into account (explicit knowledge) from those that are not necessary or already shared (implicit knowledge). Thus, a contextual system can help the user to assemble information between assets.

It is important to understand that Brézillon and Pomerol speak of knowledge because they consider context in reference to the knowledge of human actors, and here we are speaking of information exchanged between actors and contextual knowledge of the actor must be considered.

In this paper, we present the modeling of contextual information to improve the user software reuse experience in matching unlike software assets of different levels of abstraction. Furthermore, we define what is relevant or not to search and retrieve for the user based on the task at hand and on common context of different roles.

Hereafter the paper is organized in the following way. We start with conceptual definitions of software reuse (Section 2) to define the scope and domain we are working on. Section 3 presents previous works of the authors in the fields of software reuse. We then present the system in Section 4, related work is detailed in Section 5 and our conclusions and future work are given in Section 6.

2 Background

2.1 Software Reuse

Software reuse development can basically be divided in two different approaches. Development with reuse and development for reuse. The first means that one will develop its software based on artifacts previously used in the development of another software. The latter means the development of processes, tools, techniques and reusable artifacts for later development of software with reuse.

The expected benefits of software reuse are higher productivity, because the artifacts will not be built again but reused from previous projects. Higher quality, since reusable artifacts might have been previously tested and/or inspected.

Our approach is basically development for reuse. Where the software reuse repository will be used to store information and artifacts. These artifacts will

be later used to develop new software, however, as the amount of information in the repository grows, it is important to present only relevant information for each kind of user. Thus the application will need to support the creation of different roles based on company hierarchies and/or on software engineering roles to present different products to different users with different views related to which information is relevant to each one.

2.2 Context Definition

Context is a widely used concept for different areas such as psychology, linguistic or artificial intelligence, but our focus here is based and motivated by a human-centered approach described by Brézillon in [4]. This approach helped us on the focus and scope of the context modeling and also in the definition of the contextual elements. In this case, based on the context related to the user role.

Brézillon and Pomerol [5] defines context as the collection of relevant conditions and surrounding influences that makes a situation unique and comprehensible. On the other hand they also present the problem in which numerous interacting factors that people do not even pay attention to on a conscious level, and many of which are outside the ability of machine input devices to capture.

In order to computationally treat context, it is important to make this distinction between contextual data, contextual information and contextual knowledge. Therefore, we use the term contextual element (CE) to refer to pieces of data, information or knowledge that can be used to define the context. Contextual data is the basic, atomic part of the context that can be acquired directly through virtual or physical sensors, such as location coordinates, people identification or weather temperature. Contextual information is the CE that can be derived from several contextual data through association. While the information is something that once inferred can be easily instantiated and shared between human and software agents, the contextual knowledge is personal and it is inside people's head as mental schemas that help them to interpret external events.

The focus specifies what must be contextual knowledge and external knowledge, i.e. a focus in the user roles will drive the contextual knowledge about a software development project, so the implementation elements like source code programming language might be contextual knowledge for a software developer but financial information about this project might be external knowledge for him and contextual knowledge for a project manager. Although, the focus is not static, it is interlocked with its context and evolves along the execution of a series of actions resulting from the decision making process that it follows.

In summary the contextual knowledge itself has a sub-set that it is proceduralized for addressing specifically the current focus. We call it the proceduralized context. The proceduralized context is a sub-set of contextual knowledge that is invoked, assembled, organized, structured and situated according to the given focus and is common to the various people involved in decision making.

To sum up, contextual knowledge is all the contextual information that is related to a defined task, even if the information is relevant or not to the focus. External knowledge is only the information that is not the relevant for the task

or the individual. And the proceduralized context is a part of the contextual knowledge which is invoked, structured and situated according to a given focus. So, we could say that the contextual knowledge is useful to identify the activity whereas the proceduralized context is relevant to characterize the task at hand.

We cannot speak of context out of its context. Context surrounds a focus (e.g. the decision making process or the task at hand like assembling two different software artifacts) and gives meaning to items related to the focus. Thus, context guides the focus of attention, i.e. the subset of common ground that is pertinent to the current task. Indeed, context acts more on the relationships between the items in the focus than on items themselves, modifying their extension and surface. Moreover, the focus allows identifying the relevant elements to consider in the context. To help in this identification, we defined that the focus of this work would be to model contextual elements for software reuse based on the user role instead of in the artifacts. So we could have a focus of the tasks that would be executed and which contextual elements would be relevant for each role.

3 Previous Work

In this section we give an introduction to previous works developed by the authors related to software reuse and context. These previous works complement each other and are joined in the next section where we present the system.

3.1 Software Reuse

The RiSE framework [2] main objective is a systematic software reuse adoption. It has been validated in many areas sucha as software component certification, software reuse process, software reuse metrics, domain analysis, software architecture evaluation and software component search and retrieval [6].

As stated in [7], an efficient search engine should consider among other requirements the active search or proactive one. In this context, a first proactive search approach was proposed in [8] but this work did not focused on an important requirement like context information.

In this paper we consider the inclusion of context information in our search engine called B.A.R.T. (Basic Asset Retrieval Tool) project to reduce the problem identified by Frakes [9], that he identified as one of the main problem of software reuse is the "*no attempt to use*" where the user not even tries to reuse, because he is not aware of the possibility of something reusable is available for him.

B.A.R.T Project The challenge for researchers developing programming tools and environments for high-performance computing is to enable application programmers to more easily develop software systems that exploit contemporary architectures, while scaling up through the physical aspects of the problem, including problem size, data set size and complexity, the coupling of component solutions, and the complexity of numerical calculations [10].

Through the years, a vast collection of tools have been prototyped. Some of these have been developed for integrated environments, some can cooperate loosely with some others and many are freestanding [10]. Each tool or environment is still highly specific to particular context.

According to the idea that reuse can be performed in a systematic way [11], supported by an environment to aid in the software development process activities, we constructed *B.A.R.T* (*Basic Asset Retrieval Tool*). The main idea of *B.A.R.T* project is that the environment evolves in a incremental and systematic way [11], across the whole software development life cycle phases, through the integration of different techniques and methods that act to improve the effectiveness and the results of the environment. As a result, we expect to progress towards the adoption of a systematic software reuse plan. And to support the evolution of this system we aim to adopt a context management solution to go a step further.

3.2 Context Management

Context management involves the definition of models and systems to assist the acquisition, manipulation and maintenance of a shared repository of contextual elements (CE), thus enabling the usage of these elements by different context-sensitive systems. The main idea is to reduce the complexity of building context-sensitive systems, by transferring tasks related to CE manipulation to an intermediate layer. In this light, it includes the definition of a representation model to describe and share CE sets, an infrastructure to detect, update and query CE sets, mechanisms to reason, infer and process new CE sets from existing ones, and mechanisms to identify the ICE in a focus [12].

The context management process comprises the following steps: (i) acquisition of the CEs associated to a situation from virtual or physical sensors, user interfaces (e.g. forms), persistent databases, etc. (ii) to process the acquired CE through reasoning and associations the system must use knowledge bases, and inference engines. (iii) The interpreted context is used to infer information and to trigger services that must be provided and executed.

CEManTIKA Project Vieira et al. [12] presented a context management system, named CEManTIKA (Contextual Elements Management Through Incremental Knowledge Acquisition), which proposes the incremental acquisition of contextual elements according to the usage of the context-sensitive system. CEManTIKA addresses two main issues: (1) define and manage as much contextual elements as possible in the application domain; (2) identify how to use these contextual elements to assist a specific situation distinguishing the set of relevant contextual elements.

Context is a dynamic construction that evolves with the focus. As the focus changes, the set of contextual elements that must be considered changes accordingly. So, CEManTIKA manages the different focus in the domain and, for a given focus, identify which CE Sets must be considered and instantiated to

support the task at hand (the ICE Set). A Proceduralized Context Base (PCB) maintains historical cases of the ICE Set built and their respective focus. The historical ICE Sets stored in the PCB aids the identification of the relevant CE in other focus.

In this joint we use CEManTIKA as an intermediate layer to support the inclusion of the context-sensitive features into B.A.R.T. The first step in this process is to identify the contextual elements involved in the software reuse domain, building a contextual elements base (CEB) that is used as the input for CEManTIKA. In a given focus CEManTIKA uses the CE stored in the CEB to build the corresponding PC that will support B.A.R.T. in the assets search and retrieval.

4 Description of The System

4.1 Archictecture

In this section, we present the proposed architecture to join the benefits the requirements and functions of both the asset manager and the context manager.

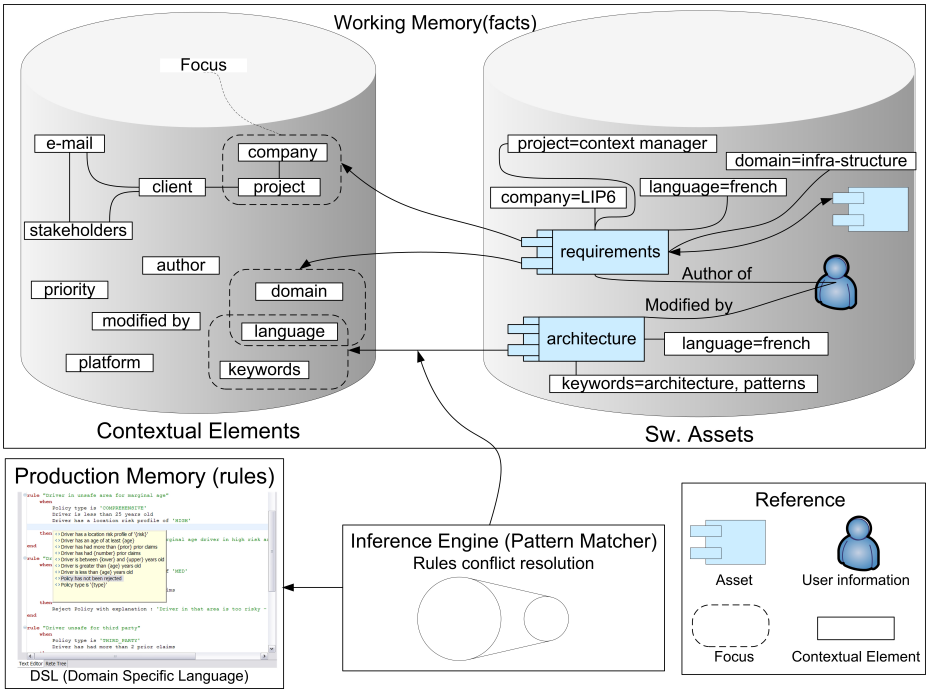

Fig. 1. Proposed Architecture

In this architecture we defined the *Working Memory* as the source of information. This will be the source of the proceduralized context.

In this source we include the contextual elements and asset elements as described in FIGURE 1. We also have the contextual elements disposed and stored together so they can be assembled and linked and are available to be instantiated as proceduralized context. On the other hand, we have *Rules* that will treat the contextual and reusable information. These rules will be described in a Domain Specific Language developed to use the users vocabulary to make easier to learn. Finally the inference engine which makes the pattern matching between the rules defined and the information in the short term memory and is also responsible for resolving possible conflicts between predefined rules.

4.2 User Role Based Context

The context manager as a tool alone is not enough to resolve the problem of contextual asset management. We need to model the contextual information to make it usable and adaptable to be useful in different aspects of the problem.

In our approach, we centered the model in the concept of the role because the notion of role is attached to each enterprise it's culture and model of work. Each enterprise is organized in terms of role. Usually a set of tasks and responsibilities are attached to each role. One or more roles are allocated to a person (an employee, an actor). This is a very important issue because the adaption of the system to the same person acting with different roles is crucial to the different tasks a user has to accomplish.

Role, task and actor are associated with each context. The item organization gives a dependency graph on the different contexts. Each of these contexts are like a filter on the domain knowledge (the contextual elements). In short, to extract the necessary contextual elements we used the perspective of the user identifying the main roles that would interact with the system in order to extract the relevant information for each one which are describe below.

4.3 Identified Roles

The identified roles were extracted from common software development life cycle roles like Rational Unified Process. We have identified: **Project Manager**, **Software Architect**, **Software Tester**, **Software Developer**, **Configuration Manager** and **Software Quality Engineer**. Our intention is not to define all possible roles but to list some common ones that make possible to model the context to help in software reuse tasks.

Note that an actor can have multiple roles associated to him and this is more common in smaller projects.

Project manager - The project manager is the role responsible for managing the project, here we use the definition of the Project Management Group (PMI), where the responsibilities of the project manager are basically related to control the time, cost and scope of the project. Many artifacts need to be controlled and revised by the project manager. Useful reusable assets for this role

would be project plans templates or project plans from similar projects where cost and time metrics could be compared.

Software Architect - This technical role is responsible for the high level structure, technology, modeling and implementation of the system. For this role architectural patterns, software requirements with functional and non-functional requirements and project test plans, and also code.

Configuration Manager - The configuration manager (CM) is the role related to the control of changes in the system. For the CM, all artifacts are relevant because he needs to track changes in any trackable artifact of the system. But for reuse purposes he would be concerned with a software configuration management plan and the project plan that also can include software configuration management information that can be reused between projects.

Software Quality Engineer - Software Quality Assurance is an issue that should concern every person in the project team, but the role that is specific related to it is the Software Quality Engineer. The actor having this role is worried about quality related activities being performed effectively. And possible reusable relevant artifacts for him are the Software Quality Plan and the project management.

Software Developer - The Software Developer is responsible for the implementation of the system. Low level technical documents, source code, components and use case documents can be reusable by software developers.

Test Engineer Test Engineers are focused in the software correctness, completeness, security and quality. And to reduce they're work and rework artifacts from previous projects like test cases and test plans can be reused by them in new projects.

4.4 Examples

The roles listed above give us the focus we need to model the context in a human centered approach. To do this, we need to think about the possible tasks at hand.

Using context to refine the search - For example, a new project manager is assigned for a project and the company has already developed other system on the same domain. In this context the project manager can have access to the previous project plans where he can see information about team sizes, costs and time to deliver versions for the client. It is expected that he will reuse this knowledge to compose his project plan with more quality than he would do with no knowledge of previous projects.

To make the relationship between the assets the user would need in this case, we need the user role to reduce the search scope and we also need other contextual elements like domain of application, project manager associated, company and sometimes even software development process can be used to assemble different project plans and retrieve then to the user even if he does not make a search, only by matching the contextual elements of his artifact with similar ones. All these examples of contextual elements can be used to match spefic contextual to refine the search and reduce the recall of assets to get a better precision.

Using context to relax the search - In a different situation a software architect who is designing the software architecture of his project identifies bottleneck points in the system. To help him in this task, a search can be made using his role as one of the contextual elements and bring to him software architecture plans of similar projects or even architectural patterns that where previously described with contextual elements like domain, kind of problem is solves or even listed as a common pattern used in the enterprise. These patterns can be related to documented non-functional software requirements or use case documentation. Based on the new architecture patterns selected use case documentation modification can be proposed, based on the documentation retrieved. In this case we use the contextual elements not to refine the search and reduce the recall but to improve it and retrieve more assets because we retrieve not only the the ones that matched the query, but those which are related to the same context.

These are examples of situations where the contextual information might be used to assemble specific contextual elements which might represent a specific contextual attribute of the asset, the task or the user. In simpler terms it might also be used to assemble a composition of contextual elements that represent a given situation or context of a specific user. For example, if we use a composition of contextual elements where the role of user is test engineer, the domain of application is games, and

4.5 Contextual Elements

Contextual elements rely on the domain knowledge, and the domain knowledge rely on (for partionning purpose) on tasks and the notion of role that control tasks. Here we list contextual elements for our domain of knowledge. They are organized and selected according to the interests of the defined roles. Depending on the focus these contextual elements will be part of the external knowledge or if they are relevant for that focus and are instantiated for a given artifact they will be part of the proceduralized context. This information is essential for the system. How we model it, stored it and relate the contextual elements with the software assets. In our approach, we related the possible contextual elements with the user role, and these roles during the software development life cycle works with different kinds of assets in different levels of abstraction. Therefore, we grouped and defined these abstraction levels in FIGURE 2 as: abstract level, design level, implementation level and management level.

The first category called *management level* are assets that are not only technical but for management purposes they need to store information about different points of view like time constraints and cost of the team in a month, they are not directly related to reusable technical artifacts the can be reused to make useful the knowledge of previous projects and also save time and improve quality of new projects. Information about time constraints and experiences with software development processes and how they where mitigated in can be extremely useful. Some assets that we can include here are project plans, quality assurance plans, software configuration management plan or even cost analysis of the project.

	ASSET TYPES LEVELS															
	Abstract			Design				Implementation						Manager		
	UML Model (Use case)	Sw. Requirements	Use Case	UML Model (Sequence)	Architecture Patterns	Test Plan	Design Pattern	UML Model (Class)	Code Snippet	Unit Test Cases	Webservice	Component	Documentation	Project Plan	quality Assurance plan	SCM Plan
Context Attributes (Common)																
company	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
project	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
client	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
license	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
domain	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
artifact type (ie. Code snippet)	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
path	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
keywords	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
author	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
language (ie. french)	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
direct connection	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
lifecycle phase (ie. Design, test)	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
stakeholders (ie. Manager, programmer)	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
version	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
security information																
visibility	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
confidentiality	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
historical Information																
creation date	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
last modification	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Context Attributes (Abstract)																
Priority		x	x													
Functional/Non-Functional		x														
Context Attributes (Design)																
Context Attributes (Implementation)																
document type (ie. Java code)								x	x	x	x	x				
programming language (ie. C#)								x	x	x	x	x				
lines of code									x	x	x	x				
class name								x	x	x	x	x				
n. of interfaces								x	x	x	x	x				
number of methods								x	x	x	x	x				
method signatures								x	x	x	x	x				
Metrics																
cyclomatic complexity									x	x	x	x				
bugs per line (1k)									x	x	x	x				
depth of inheritance tree (DIT)								x	x	x	x	x				
Textual Documents																
number of pages		x	x		x	x	x							x	x	x
glossary terms		x	x		x	x	x							x	x	x
bibliographic reference		x	x		x	x	x							x	x	x
Diagrams																
Diagrams	x			x				x								

Fig. 2. Contextual Elements Matrix

For us, *abstraction level* assets are those created in the beginning of the software development to describe the system in a high level and which will be

used later to refine other system description. In this group we include software requirements documents with it's functional and non-functional requirements. Use case documents and UML use case diagrams.

Design level assets usually receive information from high abstraction level assets, refine their information and will be used later in the implementation level. We include here UML diagrams like deployment and sequence, documentation about architecture patterns, design patterns or database diagrams. Project plans that describe how the system will be tested in a high level of abstraction are included here. Nevertheless, detailed information like implementation test cases are described are in the implementation level assets.

Finally, *implementation level* assets are those with more technical details or the executable artifacts itself. They are UML class diagrams, executable code like object oriented classes or even parts of then we call *code snippets*, documented test cases or implemented unit tests, components, or documentation about APIs.

Have made the main asset type levels we also included subcategories to identify the relationship between all these contextual elements and how we could assemble them. Starting with common attributes we can assemble any asset by then no matter how it is structures and their abstraction level. For example, a simple information like what is the project related to that asset can be used to define a context of the whole project and a user with permission access to it can list can track different abstraction levels assets based on its project.

On the other hand, we have specific contextual elements in each abstraction level, this is useful when you want for example work with a user role like software developer and in the implementation level of abstraction, in this context of use, the task at hand might be retrieve assets with a contextual element like programming language, which is specific of this kind of abstraction level and won't be found in assets like the project plan. With this concepts in mind we identify which contextual element is related to each abstraction level. Where the abstraction level is one component of the user task. But can have contextual elements that are common to any level or specific ones. Depending on the need of these elements this can be used to increase the recall of assets when we bring assets related to each other based not on specific information of search but in implicit or explicit contextual information, but in the other hand, we can also use these contextual elements to restrict and decrease the search to specific abstraction levels or roles.

5 Related Work

In this section we present some related work related to formal specification, organizational learning and we show similarities and differences from our approach.

5.1 1996 - KACTUS

KACTUS [13] stands for modelling Knowledge About Complex Technical systems for multiple USe. It is an European ESPRIT-iii project aiming at the development of a methodology for the reuse of knowledge about technical systems

during their life-cycle. This implies using the same knowledge base for design, diagnosis, operation, maintenance, redesign, instruction, et cetera. Reuse is be achieved by giving these knowledge bases an explicit structure (often called an ontology). Our approach does not use an ontology based solution in order to reduce the effort to introduce the solution and also reduce the need of an expert who would model the ontology.

5.2 1997 - CBR

In *Managing Software Engineering Experience for Comprehensive Reuse* [14] the authors introduce a tool architecture supporting continuous learning and reuse of of experience from the software engineering domain. Such retrieval is realized using context-sensitive queries and similarity functions based on case-based reasoning technology. While their approach is focused in cases and learning experience, ours has the same objective of help the user learn and finish the task at hand, but we are in artifact retrieval.

5.3 1999 - Formal Deduction Based

In 1999 Baar [15] used an approach called deduction-based software component retrieval. This approach uses formal specifications as component indexes and as queries, builds proof tasks from these, and checks the validity of the tasks using an automated theorem provers (ATP). A component is retrieved if the prover succeeds on the associated task-retrieval becomes a deductive problem. The problem with this approach is that using a formal method would increase the effort to model the problems and contexts and tasks at hand. Out approach gives the flexibility to use an expert to define and specify which contextual elements would compose the focus of an specific context making the process a lot more flexible.

5.4 2001 - CodeBroker

In 2001 Ye, implements the *CodeBroker [16]* project. CodeBroker is a proactive search tool wich context-aware browsing. Unfortunatelly, Ye proposes the architecture of the tool but does not define how the information must be modeled, which contextual elements must be used and which roles would be interacting with the system. Also the Codebroker is only focused in source code retrieval.

5.5 2005 - Strathcona

Holmes presents *Strathcona [17]* which is not a proactive search engine, but used the concept of structural context, based on java source code relations and dependencies to define relationship between the user activity and the source code stored in a repository. But Strathcona uses only the structural information of the source code to make the search and retrieval. Meaning that different actors, with different roles, executing different tasks, are considered the same way.

6 Conclusion and Future Work

Here we presented the evolution and merge of two fields, software reuse and context management. Software reuse is the domain we are working on as a problem to solve and context management a complementary field that we use to improve the solutions in a human centered approach. As described, it is not common to use context neither to enrich the semantics of the software artifacts stored in the repository nor to improve the possibilities of assembly between assets for further retrieval. In brief, we described how to use context to improve the semantics of the software assets stored in the repository in a manner that these context-enriched assets have more semantic information and we can use this information to related each other and propose useful artifacts for the user´s task at hand. On the other hand, we use the same approach to reduce the effort the user would need to search for a specific asset, because we can use not the explicit and conscious information he uses for a search query, but also the context of the user role, their task at hand and interacting factors that people do not even pay attention to on a conscious level as described by Brézillon and Pomerol [5].

We believe that this approach is very relevant and useful for the user and as future work we intend to make a formal collaboration between the Laboratoire d'informatique de Paris 6 (France) and the Informatics Center from Federal University of Pernambuco (Brazil) in order to go on with further research in this field. The project scope will be to continue the study of the application of context to asset management, definition of the necessary services the context-aware repository will need to have, implement the solution, validate in industrial environment and generate the evaluation reports.

As a result, we expect to improve the expertise in software reuse, context and software development quality and productivity of the partners. Exchange knowledge between institutions. Develop a product that will help the asset management of the institutions. As a parameter we expect also this project to be as relevant as the DEC VAX project caller R1/XCON (eXpert CONfigurer) with has been used with success in the eighties to save effort and money from DEC, using production rules to reduce hardware assemble errors of their sales orders. In our case, the results are expected in the software level, reducing the effort and improving the quality to produce, adapt or maintain software.

We also expect for an specification to develop an Asset Configurator at the software level for an optimal configuration based on technical internal software viewpoint and also from the user viewpoint adapted to his specific needs. Furthermore, commercial partnership with industry is also expected to result in sales of a final product.

References

1. McIlroy, M.D.: Mass produced software components. In: NATO Software Engineering Conference, Garmisch,Germany (1968) 138–155
2. de Almeida, E.S., Alvaro, A., Lucrédio, D., Garcia, V.C., de Lemos Meira, S.R.: Rise project: Towards a robust framework for software reuse. In Zhang, D.,

Grégoire, É., DeGroot, D., eds.: IRI, IEEE Systems, Man, and Cybernetics Society (2004) 48–53
3. P., B., J, P.: Contextual knowledge sharing and cooperation in intelligent assistant systems. Le Travail Humain **62**(3) (1999) 223–246
4. Brézillon, P.: Focusing on context in human-centered computing. IEEE Intelligent Systems **18**(3) (2003) 62–66
5. Brézillon, P., Pomerol, J.C.: Some comments about knowledge and context (2001)
6. Garcia, V.C., Lucrédio, D., Durão, F.A., Santos, E.C.R., de Almeida, E.S., de Mattos Fortes, R.P., de Lemos Meira, S.R.: From specification to experimentation: A software component search engine architecture. In Gorton, I., Heineman, G.T., Crnkovic, I., Schmidt, H.W., Stafford, J.A., Szyperski, C.A., Wallnau, K.C., eds.: CBSE. Volume 4063 of Lecture Notes in Computer Science., Springer (2006) 82–97
7. Lucrédio, D., do Prado, A.F., de Almeida, E.S.: A survey on software components search and retrieval. In: EUROMICRO, IEEE Computer Society (2004) 152–159
8. Mascena, J.C.C.P., Garcia, V.C., de Almeida, E.S., Meira, S.R.L.: Admire: Asset development metrics-based integrated reuse environment. In: XX Brazilian Symposium on Software Engineering. (2006)
9. Frakes, W.B., Fox, C.J.: Quality improvement using a software reuse failure modes model. IEEE Trans. Software Eng. **22**(4) (1996) 274–279
10. Harrison, W., Ossher, H., Tarr, P.: Software engineering tools and environments: A roadmap. In: 22nd International Conference on on Software Engineering, Future of Software Engineering Track, Limerick Ireland, ACM Press (2000) 261–277
11. Frakes, W., Prieto-Diaz, R., Fox, C.: DARE: Domain analysis and reuse environment. Annals of software engineering (1998)
12. Vieira, V., Tedesco, P., Salgado, A.C., Brézillon, P.: Investigating the specifics of contextual elements management: The cemantika approach. In: Context'07 (waiting for approval). (2007)
13. Schreiber, A.T.: The kactus booklet version 1.0. esprit project 8145. september, 1996. Technical report (1996)
14. Althoff, K., Birk, A., Tautz, C.: The experience factory approach: Realizing learning from experience in software development organizations (1997)
15. Thomas Baar, Bernd Fischer, D.F.: Integrating deduction techniques in a software reuse application. Volume 5., J. UCS (1999) 52–72
16. Ye, Y., Fischer, G.: Context-aware browsing of large component repositories. In: ASE, IEEE Computer Society (2001) 99–106
17. Holmes, R., Murphy, G.C.: Using structural context to recommend source code examples. In Roman, G.C., Griswold, W.G., Nuseibeh, B., eds.: ICSE, ACM (2005) 117–125