Towards a Software Developer Context Model

Bruno Antunes*, Francisco Correia and Paulo Gomes

Knowledge and Intelligent Systems Laboratory
Cognitive and Media Systems Group
Centre for Informatics and Systems of the University of Coimbra
Coimbra, Portugal
{bema,fcorreia,pgomes}@dei.uc.pt
http://www.cisuc.uc.pt/

Abstract. The context of a developer can be viewed as a rich and complex network of elements across different dimensions that are not limited to the work developed on an IDE. We propose the definition of a software developer context model that takes into account all the dimensions that characterize the work environment of the developer. We are especially focused on the project dimension and present a definition of what the context model encompasses at this level. The experimental work done on the context capture perspective show that useful contextual information can be extracted from project management tools. The extraction, analysis and availability of these contextual information can be used to enrich the work environment of the developer with additional knowledge that relate with the tasks being developed.

Key words: Context, Software Development, Knowledge Management.

1 Introduction

The term context has an intuitive meaning for humans, but due to this intuitive connotation it remains vague and generalist. Furthermore, the interest in the many roles of context comes from different fields such as literature, philosophy, linguistics and computer science, with each field proposing its own view of context [1]. The term context typically refers to the set of circumstances and facts that surround the center of interest, providing additional information and increasing understanding.

The context-aware computing concept was first introduced by Schilit and Theimer [2], where they refer to context as *"location of use, the collection of nearby people and objects, as well as the changes to those objects over time"*. In a similar way, Brown et al. [3] define context as location, identities of the people around the user, the time of day, season, temperature, etc. In a more generic definition, Dey and Abowd [4] define context as *"any information that can be used to characterize the situation of an entity. An entity is a person,*

* Supported by FCT grant SFRH/BD/43336/2008.

place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves".

As stated by Mena et al. [5], it is clear the importance of context for modeling human activities (reasoning, perception, language comprehension, etc), which is true in social sciences as in computing. But a generic context definition is hard to achieve, as context assumes different definitions and characteristics according to the domain where it is used. Nevertheless, there are a set of invariant characteristics in the different definitions of context: context always relates to an entity; is used to solve a problem; depends on the domain of use and time; and is evolutionary because it relates to a dynamic process in a dynamic environment.

Especially in software development, the context of a developer can be viewed as a rich and complex network of elements across different dimensions that are not limited to the work developed on an IDE (Integrated Development Environment). Due to the difficulty on approaching such challenge, there is not a unique notion of what it really covers and how it can be truly exploited. One of the problems software developers face today is the increasing dimension of software systems. Software development projects have grown in complexity and size, as well as in the number of functionalities and technologies involved. During their work, software developers need to cope with a large amount of contextual information that is typically not captured and processed in order to enrich their work environment.

Our aim is the definition of a software developer context model that takes into account all the dimensions that characterize the work environment of the developer. We propose that these dimensions can be represented as a layered model with four main layers: personal, project, organization and domain. Also, we believe that a context model needs to be analyzed from different perspectives, starting with context capture, going through its modeling and representation and ending up with its use and application. This way, each layer of the proposed context model will be founded in a definition of what context capture, modeling, representation and application should be for that layer.

This work is especially focused on the project layer the software developer context model. We give a definition of what the context model encompasses at the project layer and present some experimental work on the context capture perspective.

The remaining of the paper starts with an overview of the software developer context model we envision. In section 3 we describe the current work and some preliminary experimentation is discussed in section 4. An overview of related work is given in section 5. Finally, section 6 concludes the paper and point out some directions for future work.

2 Software Developer Context Model

The software developer context model we propose take into account all the dimensions that comprise the software developer work environment. This way, we have identified four main dimensions: personal, project, organization and do-

main. As shown in figure 1, these dimensions form a layered model and will be described from four different perspectives: context capture, context modeling, context representation and context application. In the following sections the context model layers and perspectives are described in more detail.

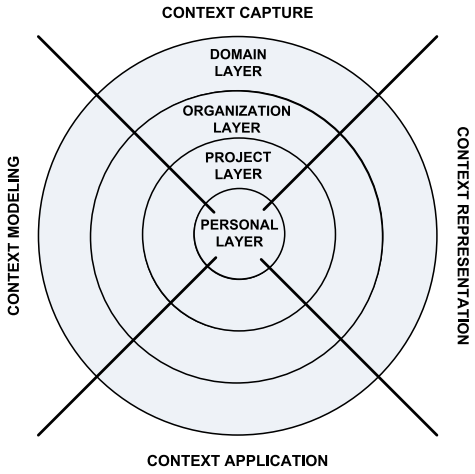

Fig. 1. The software developer context model layers and perspectives.

2.1 Context Model Layers

The developer context model we envision is based on a layered model made up of four layers: *personal layer*, *project layer*, *organization layer* and *domain layer*. Each one of these layers focus on different dimensions of the environment where a typical developer works. In the following sections we describe what is perceived as contextual information in each one of these dimensions.

Personal Layer. The personal layer represents the context of the work a developer has at hands at any point in time, which can be defined as a set of tasks. In order to accomplish these tasks, the developer has to deal with various kinds of resources at the same time, such as source code files, specification documents, bug reports, etc. These resources are typically dispersed through different places and systems, although being connected by a set of explicit and implicit relations that exist between them. At this level the context model represents the resources that are important for the tasks the developer is working on. For instance, when the developer is working on a specific task, there are a set of resources that are more relevant for that task than others and should be highlighted to the developer to facilitate her/his work.

Project Layer. The project layer focuses on the context of the project, or projects, in which the developer is somehow involved. A software development project is an aggregation of a team, a set of resources and a combination of explicit and implicit knowledge that keeps the project running. The team is responsible for accomplishing tasks, which end up consuming and producing resources. The relations that exist between people and resources are the glue that makes everything work. The project layer represents the people and resources, as well as their relations, of the software development projects where the developer is included. For instance, when fixing a specific bug it is important to know what other bugs might be related, which files are likely to be affected and which other developers working on the project may be of help to solve the bug.

Organization Layer. The organization layer takes into account the organization context to which the developer belongs. Similarly to a project, an organization is made up of people, resources and their relations, but in a much more complex network. While in a project the people and resources are necessarily connected due to the requisites of their work, in a software development organization these projects easily become separate islands. The knowledge and competences developed in each project may be of interest in other projects and valuable synergies can be created when this information is available. The organization layer represents the organizational context that surrounds a developer. For instance, when a developer needs to apply a specific technology with which no one of the project team is comfortable, there may be other colleagues, or even resources, in the same organization which can help.

Domain Layer. The domain layer takes into account the knowledge domain, or domains, in which the developer works. This layer goes beyond the project and organization levels and includes a set of knowledge sources that stand out of these spheres. Nowdays, a typical developer uses the Internet to search information and to keep informed of the advances in the technologies s/he works with. These actions are based on services and communities, such as source code repositories, development foruns, news websites, blogs, etc. These knowledge sources cannot be detached from the developer context and are integrated in the domain layer of our context model. For instance, due to the dynamic nature of the software development field, the developer must be able to gather knowledge from sources that go beyond the limits of the organization, either to follow the technological evolution or to search for help whenever needed.

2.2 Context Model Perspectives

In the following sections, we present a set of different perspectives that apply to each one of the context model dimensions discussed before: *context capture*, which represents the sources of context information and the way this information is gathered, in order to build the context model; *context modeling*, which represents the different dimensions, entities and aspects of the context information

(conceptual model); *context representation*, which represents the technologies and data structures used to represent the context model (implementation); and *context application*, which represents how the context model is used and the objectives behind its use.

Context Capture. The context capture process is in the basis of any context-aware process. The quantity, quality and coverage of the context information available influences the definition of the context model at a first stage and the quality of the results in a second phase. Depending on the approach followed, this process can also be influenced by a predefined context model, which ultimately determines the context information needed. Also, the context gathering process is highly dependent on the domain and strongly limited by the environment where the user develops her/his activities.

In this work, we are concerned with the context information available in the work environment of a developer and how it can be retrieved. Most of the working time of a developer is spent in an IDE, this is the scenario where most of the action takes place, thus it is where a great amount of contextual information is available. But, the work of a developer is not limited to the use of an IDE, s/he typically needs to deal with information that is provided by different applications and systems, interact with other developers, search for information in sources inside and outside the organization scope, etc. While most of the approaches studied focus on the IDE as the only way to observe and get feedback from the developer, we believe that although the IDE represents the privileged way to get in touch with the developer, there are other paths that should not be ignored.

Context Modeling. The context model represents a definition of what context is in a specific domain. This definition must take into account the different dimensions on which context is present, as well as the various elements that pertain to each dimension. Again, the definition of a context model is highly dependent on the domain where it applies and may be very different from one application to another. It does not only depend on the characteristics of the domain, but also on the level of complexity needed in order to achieve the objectives of the application. Furthermore, it is somehow subjective, in the sense that it represents a specific perspective of what context is in a specific domain and may not be the only perspective available.

The definition of a context model for the software development domain is one of the main objectives of this work. Although some of the works studied already define some kind of context model, they commonly focus on what we consider to be a small portion of a bigger picture. This commonly happens because the objective was not to create a global model that could represent the context of the developer, but instead a context model that could aid a specific objective. Also, as discussed in the previous section, the working environment of the developer is commonly confined to the IDE, which we believe to be a reductive perspective on what the work of a developer usually comprises. We aim to create a unified context model that integrates all the dimensions of the context

information provided by the working environment of a developer, as described before when discussing the layered architecture of the developer context model.

Context Representation. The context representation deals with the way context information is actually represented and depends on the complexity of the context model. The technology or data structure used must be expressive enough to represent the details of the context information and suitable to deal with the needs of the application regarding the use of that information.

The approaches studied present various ways of representing context information, ranging from simple lists to ontologies [6]. The simplest structures are easier to build and maintain, but provide limited reasoning over the context information, thus limiting the outcomes we may expect from such approaches. On the other hand, semantically rich structures, such as ontologies, provide a more powerful formalism to store and exploit context information, but are more difficult to build, maintain and validate. Ultimately, the complexity of the context model is what most determines the choice of what technology or structure to use.

Taking into account various issues, we propose to use one or more ontologies in order to represent the context model we envision. First, we think that the complexity of a global and unifying context model for software development demands the use of a complex structure, such as an ontology. Although some approaches already make use of ontologies to represent their context model, we believe that they are under-explored, especially in the software development domain. The ontologies can be represented and manipulated using the Semantic Web technologies. We believe that the use of the Semantic Web technologies is an advantage, first because of the standards accepted by the scientific community and second because they promote the reusability and knowledge sharing.

Context Application. The ultimate objective of capturing, modeling and representing context information is always the improvement of some process and the consequent improvement on the results we may expect from that process. We focus on software development, especially from the perspective of the developer.

By taking into account the context of the developer, we are able to filter out information that does not fit the actual interests of the user, to rank information in order to reflect the actual importance they may have to the developer and to elicit information that could not be highlighted otherwise. But, where we believe we can make the difference is presenting relevant information to the user that s/he never asked for, but which may be relevant for the activity s/he is performing.

The process of suggesting information to the user is of special importance when we get out of the scope of the IDE and take into account information that is not present in such environment, but is very important for the activities of the developer. The objective is always to present the most relevant information to the developer, either when s/he explicitly demands for that information or when the system considers that information to be relevant to the user.

3 Current Work

We are currently working on the project layer of our developer context model, and we will discuss our work at this level from the different perspectives we have presented before.

Concerning context capture, the main sources of contextual information that feed up the developer context model at the project level are project management tools. These tools store a big amount of explicit and implicit information about the resources produced during a software development project, how the people involved relate with these resources and how the resources relate to each other. We are focusing on two types of tools: *Version Control Systems*[1] (VCS) and *Issue Tracking Systems*[2] (ITS). As shown in figure 2, the former deals with resources and their changes, the second deals with tasks. These systems store valuable information about how developers, tasks and resources relate and how these relations evolve over time. We are especially interested in revision logs and tasks. Briefly described, a revision log tell us that a set of changes were applied by a developer to a set of resources in the repository. A task commonly represents a problem report and the respective fix.

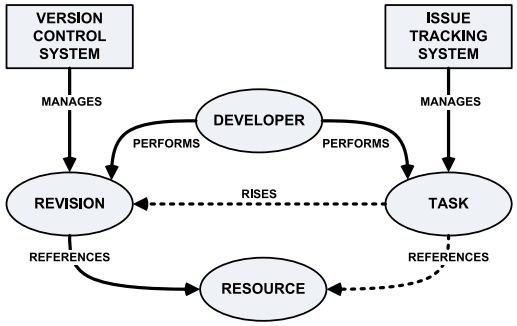

Fig. 2. The project layer relevant entities and their roles.

The network of developers, resources, tasks and their relations will be used to build our context model at the project level. This way, the context model of the developer, from a project point of view, will be modeled as a set of implicit and explicit relations, as shown in figure 3. The lines with a filled pattern represent the explicit relations and those with a dotted pattern represent the implicit ones. The developers are explicitly related with revisions, as they are the ones who commit the revisions, and with tasks, as each task is assigned to a developer. The relation between tasks and resources is not explicit, but we believe it can be identified by analyzing the information that describe tasks and revisions. The

[1] `http://en.wikipedia.org/wiki/Version_control_system`
[2] `http://en.wikipedia.org/wiki/Issue_tracking_system`

proximity between developers can be inferred by analyzing the resources were they share work. Also, the resources can be implicitly related by analyzing how often they are changed together.

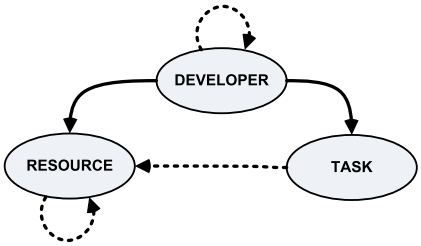

Fig. 3. The elements that compose the context model in the project layer.

In order to extract relations from the information stored in project management tools, that information is previously imported and stored locally for analysis. The prototype developed uses a database to store both the imported data and extracted relations. In the next phase, we intend to represent these relations and connection elements in an ontology, which will gradually evolve to a global developer context model ontology.

The context information extracted at the project level will be used to inform the developer about the network that links people, resources and tasks on the project s/he works. This information can be prepared to facilitate consulting and presented to the developer easily accessible in her/his working environment. This way the developer can easily gather information about what resources are likely to be implicated in the fulfillment of a task, what other tasks affected these resources in the past, and what other developers may be of help if extra information is needed.

4 Experimental Work

The experimental work conducted so far is focused on the project layer of the developer context model, which has been discussed in the previous section. We have elected VCS and ITS as two central project management tools, from the developer point of view, and started using the information provided by these tools in order to build our context model.

We have implemented connectors that allowed us to collect all the desired information from the *Subversion*[3] and *Bugzilla*[4] systems, as they are among the most popular in use. Through the collected information, we could already perceive a set of explicit relations: which resources are created/changed by which

[3] `http://subversion.tigris.org/`
[4] `http://www.bugzilla.org/`

developers, which tasks have been assigned to which developers, which resources are likely to be related because they were changed together, etc. There is also a set of implicit relations that would be valuable if disclosed: which revisions are associated with a task and which resources are associated with a task.

4.1 Relation Extraction

Our approach to extract implicit relations between revision logs and tasks relies on the analysis of the text provided by revision messages, task summaries and task comments. It is common to find references to task identifiers in revision messages, denoting that the revision was made in the context of a specific task. Also, task summaries and comments commonly reference specific resources, either because a problem has been identified in a specific class or because a error stack trace is attached to the task summary to help developers identify the source of the problem. Taking this into account, we have defined three algorithms to find resource/task and task/revision relations:

- *Resource/Task (I).* For each resource referenced in a revision, the respective name was searched in all task summaries. The search was performed using the file name and, in case it was a source code file, the full qualified name (package included) and the class name separately.
- *Resource/Task (II).* For each resource referenced in a revision, the respective name was searched in task comments. This search was performed as described for the previous relation.
- *Task/Revision.* For each task, the respective identification number was searched in revision messages. The search was performed using common patterns such as "<id>", "bug <id>"and "#<id>".

4.2 Preliminary Results

To validate the relation extraction algorithms, these were tested against two open-source projects from the Eclipse[5] foundation:

- *gEclipse*[6]. The gEclipse framework allows users and developers to access Computing Grids and Cloud Computing resources in a unified way, independently of the Grid middleware or Cloud Computing provider. Our analysis was performed over the work of 19 developers in a time window of approximately 3 years and 3 months, which included 3279 revisions and 1508 tasks with 7765 comments.
- *Subversive*[7]. The Subversive project supports the development of an Eclipse plug-in for Subversion integration. Our analysis was performed over the work of 10 developers in a time window of approximately 3 years and 2 months, which included 1120 revisions and 1013 tasks with 3252 comments.

[5] `http://www.eclipse.org/`
[6] `http://www.eclipse.org/geclipse/`
[7] `http://www.eclipse.org/subversive/`

Table 1. Number of extracted relations.

	Resource/Task (I)	**Resource/Task (II)**	**Task/Revision**
gEclipse	2527	31671	629
Subversive	208	9076	773

By applying the relation extraction algorithms to the information related with these two projects, we have gathered the results represented in table 1. The table shows the number of distinct relations extracted using each one of the algorithms in the two projects analyzed.

The results show that a large amount of implicit relations can be extracted from the analysis of the information associated to tasks and revisions. These relations complement the context model we are building by connecting tasks with related resources. With a more detailed analysis we have identified some problems with the algorithms creating some relations that do not correspond to an effective correlation between the two entities analyzed. These problems are mainly related with string matching inconsistencies and can be corrected with minor improvements in the way expressions are searched in the text.

5 Related Work

The EPOS (Evolving Personal to Organizational Knowledge Spaces) project, presented by Schwarz [7], aims to build organizational structured knowledge from information and structures owned by the elements of the organization. The world of a knowledge worker is defined as containing document-like objects, objects used for classification and applications. This work environment is taken into account when modeling the user context, which comprises information about various aspects, including currently or recently read documents, relevant topics, relevant persons, relevant projects, etc. The context model is formalized using RDFS [8], and each user's context is an up-to-date instance of this context model. The context information is gathered and modeled through user observation and/or user feedback. The gathering and elicitation of contextual information is done in two ways: context-polling, by requesting a snapshot of the user's current context; and context-listening, by subscribing the Context Service to be informed of every change in the user's context. Being a developer a knowledge worker, much of the concepts referenced in this work apply to the software development domain, but the specificities of this domain demand for a context model adapted to the reality of the work environment of a software developer.

Modularity is in the basis of the development of complex software systems and largely used to support a programmer's tasks, but not always help the programmer finding the desired information or delimit the point of interest for a specific task. Based on this, Kersten and Murphy [9] have been working on a model for representing tasks and their context. The task context is derived from

an interaction history that comprises a sequence of interaction events representing operations performed on a software program's artifact. They have developed Mylar, now called Mylyn[8], which uses the information in a task context either to help focus the information displayed in the IDE, or to automate the retrieval of relevant information for completing a task. The focus of this work is the task and the knowledge elements present in the IDE that are more relevant for the fulfillment of that task. Our approach aims to define a context model that goes beyond the IDE and explores the knowledge provided by the different systems that support the software development process.

In the same line of task management and recovery, Parnin and Gorg [10] propose an approach for capturing the context relevant for a task from a programmer's interactions with an IDE, which is then used to aid the programmer recovering the mental state associated with a task and to facilitate the exploration of source code using recommendation systems. Their approach is focused on analyzing the interactions of the programmer with the source code, in order to create techniques for supporting recovery and exploration. Again, this approach is largely restricted to the IDE and the developer interaction with it.

With the belief that customized information retrieval facilities can be used to support the reuse of software components, Henrich and Morgenroth [11] propose a framework that enables the search for potentially useful artifacts during software development. Their approach exploits both the relationships between the artifacts and the working context of the developer. The context information is used to refine the search for similar artifacts, as well as to trigger the search process itself. The context information model is represented with RDF [12] statements and covers several dimensions: the user context, the working context, and the interaction context. While the focus here is in software reuse, our approach focuses on the information captured during the process development.

6 Conclusions

We have presented our approach of a software developer context model. Our context model is based on a layered structure, taking into account four main dimensions of the work environment of a software developer: personal, project, organization and domain. We proposed that each layer should be defined taking into account context capture, modeling, representation and application and explained what should be take into account in each one of these perspectives.

The current work is focused on the project layer of the software developer context model. We have discussed this layer in more detail and presented preliminary experimentation on the context capture perspective. The results show that it is possible to relate tasks and revisions/resources using simple relation extraction algorithms. The relations extracted help us defining the network that exist between developers, tasks and resources. This network represents the context model of the developer at the project layer and can be used to unveil useful information to the developer.

[8] `http://www.eclipse.org/mylyn/`

As future work we plan to integrate the contextual information in an IDE, for instance using an Eclipse plug-in, and present this information to the developer in a way that should not be intrusive but useful for the tasks that s/he is performing. Once this information is integrated in the work environment of the developer, it is crucial to understand what her/his personal context comprises, so that the information provided can be adequate to her/his information needs. The remaining layers of the context model will be addressed iteratively, as an extent to the work already developed. Finally, we pretend to test our approach with developers working in real world projects.

References

1. Mostefaoui, G.K., Pasquier-Rocha, J., Brezillon, P.: Context-aware computing: A guide for the pervasive computing community. In: Proceedings of the IEEE/ACS International Conference on Pervasive Services, ICPS 2004. (2004) 39-48
2. Schilit, B., Theimer, M.: Disseminating active map information to mobile hosts. IEEE Network (1994) 22-32
3. Brown, P.J., Bovey, J.D., Chen, X.: Context-aware applications: From the laboratory to the marketplace. Personal Communications, IEEE 4 (1997) 58-64
4. Dey, A.K., Abowd, G.D.: Towards a better understanding of context and context-awareness. In: CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness, The Hague, The Netherlands (2000)
5. Mena, T.B., Saoud, N.B.B., Ahmed, M.B., Pavard, B.: Towards a methodology for context sensitive systems development. In Kokinov, B.N., Richardson, D.C., Roth-Berghofer, T., Vieu, L., eds.: Modeling and Using Context, 6th International and Interdisciplinary Conference, CONTEXT 2007, Roskilde, Denmark, August 20-24, 2007, Proceedings. Volume 4635 of Lecture Notes in Computer Science., Springer (2007) 56-68
6. Zuniga, G.L.: Ontology: Its transformation from philosophy to information systems. In: Proceedings of the International Conference on Formal Ontology in Information Systems, ACM Press (2001) 187-197
7. Schwarz, S.: A context model for personal knowledge management applications. In: Modeling and Retrieval of Context, 2nd International Workshop (MRC 2005), Edinburgh, UK (2005)
8. Brickley, D., Guha, R.V.: Rdf vocabulary description language 1.0: Rdf schema (2004) Published: W3C Recommendation.
9. Kersten, M., Murphy, G.C.: Using task context to improve programmer productivity. In: Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Portland, Oregon, USA, ACM (2006) 1-11
10. Parnin, C., Gorg, C.: Building usage contexts during program comprehension. In: Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC'06). (2006) 13-22
11. Henrich, A., Morgenroth, K.: Supporting collaborative software development by context-aware information retrieval facilities. In: DEXA Workshops, IEEE Computer Society (2003) 249-253
12. Miller, E., Manola, F.: Rdf primer (2004) Published: W3C Recommendation.