Supporting Shared Understanding within Distributed Enterprise Development Teams

Jessica Rubart, Stephan Müller

arvato direct services
An der Autobahn
33310 Gütersloh
jessica.rubart@bertelsmann.de
stephan.mueller@bertelsmann.de

Abstract: This paper presents a practitioners' report on supporting shared understanding within distributed development teams. Our software domain focuses on Enterprise development in the context of customer relationship management. We are applying meta use cases, software reuse through a component-based architecture, a UML-based modeling language, and groupware tools as means to support shared understanding.

1 Enterprise Development

We are concerned with the development of Enterprise solutions focusing on customer relationship management (CRM). This usually comprises the provision of services for end users as well as for service center agents. Examples for such services are customer registration, bonus management, provision of special offers, or mailings.

In the context of our projects it is crucial to keep time and budget. Usually, the projects go through a setup and a maintenance phase. While the setup phase typically lasts between 2 to 12 months, the maintenance phase runs over years. In addition, the software characteristically needs to be continuously available during the maintenance phase. Therefore, our software needs to be of high quality. Furthermore, innovation management is an important issue to us to keep our software competitive.

As the project setup and change request volumes differ significantly we often need to use external development resources. To keep to the available budget we have established several offshore development groups who are cheaper than external consultants. But our overall goal – to keep time and budget – can only be reached if all participants collaborate effectively. This is particularly difficult with changing development resources.

Effective collaboration requires shared understanding within the distributed development teams. Understanding the development process is one important issue in this context. In practice a development process usually is based on a process framework, such as the *Unified Process* (UP) [JBR99] or *eXtreme Programming* (XP) [Be99]. However, those process frameworks cannot be applied directly. Typically, they need to get tailored. Process descriptions and process patterns are usually quite general and, thus, difficult to apply in a concrete situation. Therefore, we are using *meta use cases* to document our development process.

In addition, in practice there are misunderstandings between team members. Even though they mean the same concept of an architecture or model of the system it happens that they do not understand each other properly. This is often due to the fact that there is no shared vocabulary and common understanding about the system architecture.

In summary, our approach to support shared understanding within distributed development teams is threefold:

- *Meta use cases* document our evolving development process.
- Developing a *component-based architecture* supports reuse in all projects. In addition, we are applying and developing a *UML-based modeling language* to support a shared vocabulary and model-driven development.
- State-of-the-art *groupware tools*, such as a version control system, a Wiki [LC01], and video conferencing, support communication as well as managing shared artifacts and workspaces.

The next section gives theoretical background on shared understanding. The subsequent sections describe our approach focusing on meta use cases, the component framework, and shared modeling language as those are our main contributions with respect to supporting shared understanding. Afterwards a comparison to related work is given. The paper ends with experiences and conclusions.

2 Shared Understanding

Shared understanding is very important for effective collaboration. While working on a project, team members need to develop shared understanding about the problem to solve, the requirements, and the solution, e.g. the evolving system architecture.

According to Landauer, data can be seen as the raw bits and bytes, while information is data with an interpretive context [La98]. Furthermore, knowledge is seen as information with a knower, while understanding is knowledge in perspective, in multiple contexts. Understanding is the most meaningful concept and thus the most difficult one – for humans as well as for computer systems.

As said by Clark, in order for one person to understand another there must be a "common ground" of knowledge between them [Cl96]. Clark argues that language use in conversation is a collaborative process, where participants work together to ensure that the language expression produced can be understood by all the participants. Establishing this common ground involves negotiating the meanings of words and constructing shared interpretations of them. Such a process is called *grounding*.

Figure 1 illustrates the four-stage model of basic communication according to Walsham [Wa04]. During her everyday activities Person A is acting and reflecting on the actions in her environment. This involves a range of *sense-reading* activities, e.g. interpreting sounds and events. To communicate to Person B, Person A re-presents her experiences through media, such as voice, data, text, or diagrams. This is called *sense-giving*. Person B then "reads" the representation of Person A, but might interpret the meaning quite differently. Finally, Person B is acting and reflecting in his environment influenced by Person A's representation. The challenge is how to support these sense-reading and sense-giving processes.

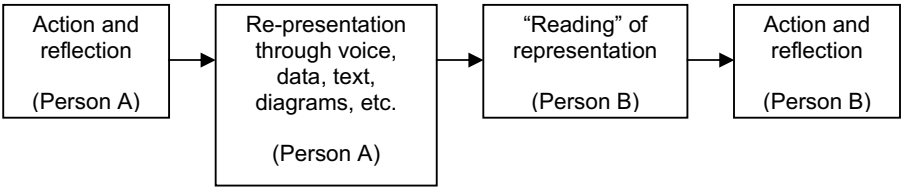

Figure 1: Four-stage model of basic communication [Wa04]

In our approach, we are applying meta use cases, software reuse through a component-based architecture, and a UML-based modeling language as means to support sense-reading and sense-giving processes. The meta use cases, the basic structure, abstractions, design patterns, etc. of the component framework as well as the shared modeling language are part of the common ground. Therefore, representations using these concepts and terms are easier to understand. Groupware tools support communication, lessons learned, and management of shared artifacts and workspaces from the technical point of view.

3 Meta Use Cases

We are using meta use cases to describe our development process. A use case is *a description of a set of sequences of actions, including variants, that a system performs that yield an observable result of value to an actor* [BRJ99]. A meta use case describes an activity of our development process focusing on roles and their responsibilities. We are executing meta use cases to implement use cases of the system to be developed.

Dependent on their roles taken in respective projects team members read relevant meta use cases and establish shared understanding about the development process. This improves effective collaboration from the process point of view. Shared understanding about the development process makes it also much easier for team members to take different roles in different projects. This increases flexibility with respect to team resource management as team members can more easily stand in for colleagues (assuming that the skills for the respective role are available).

Due to the fact that our meta use case descriptions have a similar structure as our system use case descriptions team members usually have no problem to understand them. There is no gap between the specification languages. Table 1 shows a shortened example for a meta use case description, which focuses on the specification of a system use case in the context of an offshore project. The header provides a description, a list of actors, the trigger, preconditions, results, and post conditions. In the body the activities and responsible actors are presented. The *Exception handling* area refers to and gives details on the exceptions that might occur during the meta use case, such as *Change request* in this example.

Table 1: Shortened example for a meta use case description

	Meta-UC Offshore Specify System Use Case	
Description	Specify and agree on system use case.	
Actors	SA	System analyst onsite
	OD	Operating department
	SWA	Software architect onsite
	SWA-Off	Software architect offshore
Trigger, precondition	The project manager assigns the system analyst to specify the system use case. The system analyst is familiar with the business use case. The following information is available: • Business use case • Result of very first effort estimation	
Results, post condition	The system use case is specified and approved for being developed. Change requests of the operating department might have opened the change request process. The design and development of the system use case can follow. The following results are available: • Specification of the system use case • Optional: New or adapted system interface(s) • Optional: Prototypical GUI • Optional: Analysis model	

No.	Responsible	Activity
1	**-**	**Initial description of system use case**
1.1	SA	The system analyst describes the system requirements according to the business use case by creating an initial description of the system use case. The system analyst clarifies open issues with the operating department. Dependent on the system use case the system analyst might create or adapt system interfaces, a prototypical GUI, and an analysis model.
2		**Review of the system use case**
2.1	SWA	The software architect onsite reviews and clarifies the system use case specification in the document. He might discuss clarifications with the system analyst. Additionally, the software architect can add design annotations, e.g. to suggest which available software components should be used.
2.2	SWA-Off	The software architect offshore reviews and further clarifies the system use case specification. She might discuss clarifications with the system analyst or the software architect onsite.
3		**Clearing of the system use case**
3.1	OD	The operating department reviews and approves the document(s).
3.2	OD	**Exception: Change request**
		Exception handling
3.2	OD	**Exception: Change request**
3.2.1	OD, SA	The operating department discusses the change request with the system analyst. If the change request has minimal impact the process goes on with no. 3; otherwise the process goes on with no. 2 and change request management might be necessary.
Open issues		
Comments		This meta use case does not describe the details of the system analysis.

When using or executing meta use cases we usually take into account lessons learned and the particular situation in the current project. This means that we do not necessarily execute the meta use cases strictly. Rather, we handle them more like a guideline and as a means to reflect lessons learned. In every project we are collecting such lessons learned and we consider those also in our meta use cases by adapting or extending them. Another important issue with respect to our development process is that for each project we are planning an innovation that the project should contribute to either the component framework (see next section) to keep it competitive or to our development process. Such innovations are an investment in the future of our framework and processes; each project has to make its contribution.

4 Component-based Architecture

We are applying and developing a component-based architecture, which supports reuse in all projects. Reuse-based development usually distinguishes between *development for reuse* and *development with reuse* [Ka95]. Development *for* reuse includes the development of reusable components as well as re-engineering, qualifying, and classifying those. Development *with* reuse focuses on the construction of new software as well as utilizing and, if necessary adapting, reusable components.

The scope of reuse can vary; in [Ka95] three reuse categories are distinguished:

- "General": This category refers to domain-independent reuse, such as basic user interface elements.
- "Domain": Domain-dependent components support domain reuse, e.g. customer management in the CRM domain.
- "Product-line": Product line reuse refers to reusable components that are bound to specific application types within a domain, e.g. a loyalty product-line within the CRM domain. Variability management is an important challenge in this context [PBL05].

On the one hand reuse improves efficiency as software does not need to be developed from scratch. And on the other hand, if projects are based on a common framework, reuse also improves effectiveness as there is already shared understanding about the component framework among the team members.

4.1 Component Framework

Our component framework is compliant with the J2EE standard [J2EE]. As the basis for the framework we are working with the following middleware technologies:

- PriDE as an object-relational mapper for persistency management [PriDE]
- The CUBA component framework [CUBA], which allows using the components as Enterprise Java Beans (EJBs), AXIS web services [AXIS], or in stand-alone applications without additional programming effort.

Those middleware technologies support reuse on a generic level. In addition to middleware reuse, our component framework distinguishes between the following component types:

- *Basic services* provide adapters to external systems, such as persistency services.
- *Business logic components* provide business logic utilizing basic services or other business logic components.
- *Application logic components* provide application specific logic typically utilizing business logic components.
- *GUI components* are provided using Java Beans.

While our middleware and GUI components are domain-independent, the basic services, business logic components and application logic components are dependent on a specific domain. In addition, the application logic components are application dependent. We are using the term *business component* for a specific bundle of different kinds of components that provide comprehensive support for a specific part of the domain.

4.2 Shared Modeling Language

To support a shared vocabulary among the team members and model-driven development we are using and developing a UML (*Unified Modeling Language)*-based modeling language. For this we are utilizing the concept of a UML profile. Such a profile is a lightweight mechanism to tailor the UML towards specific domains or platforms [UML].

We have developed stereotypes and tagged values to mark and configure specific component types and further important abstractions of our framework in the design model. In addition, a code generator has been developed that supports code generation where appropriate, e.g. initializing a database for modeled persistent entity classes.

To our experiences, visual modeling and the Model Driven Architecture [MDA] approach support abstraction, specific viewpoints, reuse on the model as well as software component level, portability, and readability. Therefore, it is an important means to support shared understanding. We are currently discussing model transformations for supporting the mapping of a domain model into a design model. Furthermore, we are talking about how the transformation of a business process into a technical workflow can be supported.

5 Groupware Tools

We are using state-of-the-art groupware tools, such as a version control system, a Wiki, video conferencing, or application sharing. To our experiences, such tools are particularly supportive with respect to communication as well as management of shared artifacts and workspaces. Currently, we are not using a workflow management system for supporting our development process as those tend to be too restrictive. At present, it seems easier and more flexible to work with, learn from, and adapt or extend our meta use cases. The Active Knowledge Model (AKM) approach [Li03], which has been extended in the EXTERNAL project to include interactive process models and interactive model activation [RJ04], seems promising to support flexible development processes. However, there is no commercial product available yet. But, we are in the process of setting up a collaborative project management tool that makes tracking the progress of projects easier as every team member can access and fill in shared project plans. In addition, a cooperation-aware UML tool can be an improvement to also have support for synchronous distributed design discussions using e.g. sequence diagrams. Currently, we are using application sharing for this, which has its limitations as it is basically a remote control of a single user application.

6 Comparison to Related Work

Compared to approaches in the area of knowledge management this paper takes the view of Geoff Walsham [Wa04] by not talking about "knowledge repositories" or "knowledge sharing", but means to support sense-reading and sense-giving processes.

A process pattern is seen as *a collection of general techniques, actions, and/or tasks (activities) for developing object-oriented software* [Am98]. Thus, process patterns do not describe the details of how to do something, but are written down more like best practices. Compared to process patterns and frameworks meta use cases are very concrete and easy to understand for all team members.

Workflow management systems as well as the initial AKM approach [Li03] can be used to represent meta use cases as workflow models. However, the execution support would be too restrictive. The extended AKM approach from the EXTERNAL project, which includes interactive process models and interactive model activation [RJ04], seems promising to support the execution of meta use cases. However, the representation to the end user would be challenging.

Patterns for Computer-Mediated Interaction [SL07] provide a pattern language that can be used by all stakeholders when using or developing groupware. They can serve as a shared vocabulary and, thus, as a means to support shared understanding in the computer-mediated interaction domain. From this perspective they are complementary to meta use cases.

7 Experiences and Conclusions

This is a practitioners' report on supporting shared understanding within distributed development teams. It focuses on means that we successfully apply to support sense-reading and sense-giving processes. These include meta use cases, software reuse through a component-based architecture (its structure, design patterns, abstractions, etc.), and a shared modeling language. In addition, we are applying state-of-the-art groupware tools, such as a version control system, a Wiki, and video conferencing, to support communication as well as managing shared artifacts and workspaces.

According to our experiences, a new team member needs about a week of training period to get a basic understanding about the component framework. Reusing a component framework supports shared understanding as team members can more easily identify responsible components within different projects. We also found it very useful to provide a similar set of artifacts in each project, such as a document describing the mapping of the domain model to the design model or one, which provides details about the system architecture. In practice there are misunderstandings between team members. A shared modeling language supports a shared vocabulary among the team members and, thus, also contributes to shared understanding. In addition, visual modeling and the MDA approach support shared understanding through abstraction, specific viewpoints, reuse on the model as well as software component level, portability, and readability. We also provide a model about our component framework serving as a meta model for the project-specific design models. In addition, we found it very useful to give an overview model about use cases. This also shortens the training period of new project members. As glue between use cases and components we usually utilize sequence diagrams.

Compared to process frameworks, descriptions, and patterns meta use cases are very concrete and easy to understand for all team members. Due to the fact that our meta use case descriptions have a similar structure as our system use case descriptions team members usually have no problem to understand them. In addition, to our experiences meta use cases also ease the adaptation and evolution of our development process as each of them focuses on a specific part of the process. Furthermore, meta use cases support flexible resource management as team members can more easily take different roles and stand in for colleagues. To our experiences, state-of-the-art groupware tools are particularly supportive with respect to communication as well as management of shared artifacts and workspaces. Currently, it seems challenging to find and use a process support tool for running our meta use cases in a flexible way. At present, we simply work with our meta use case descriptions and utilize lessons learned as well as a knowledge management process to improve our meta use cases. Our version control system is the main tool we are using for managing shared artifacts and workspaces. We are using a Wiki for supporting a project's maintenance phase by collaboratively collecting and keeping up maintenance documentation, such as possible problems and their solutions. Video conferencing is an important means to us for regular meetings of a distributed development team. For synchronous work on shared artifacts we are currently utilizing application sharing and shared whiteboards. More specific cooperation-aware tools can be an improvement.

References

[Am98] Ambler, S. W.: Process Patterns: Building Large-Scale Systems Using Object Technology. Cambridge University Press, 1998.

[AXIS] Apache Axis, http://ws.apache.org/axis/, http://ws.apache.org/axis2/.

[Be99] Beck, K.: Extreme Programming Explained. Addison-Wesley, Reading, MA, 1999.

[BRJ99] Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley, Reading, MA, 1999.

[Cl96] Clark, H. H.: Using language. Cambridge University Press, 1996.

[CUBA] The CUBA framework, http://cuba.sourceforge.net.

[J2EE] Java 2 Platform Enterprise Edition, http://java.sun.com/javaee/.

[JBR99] Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley, 1999.

[Ka95] Karlsson, E.-A. (Editor): Software Reuse. A Holistic Approach. John Wiley & Sons, 1995.

[LC01] Leuf, B., Cunningham, W.: The Wiki Way: Quick Collaboration on the Web. Addison-Wesley, 2001.

[La98] Landauer, C.: Data, Information, Knowledge, Understanding: Computing Up the Meaning Hierarchy. In Proceedings of SMC'98: The IEEE Int. Conf. on Systems, Man, and Cybernetics, California, 1998.

[Li03] Lillehagen, F.: The Foundations of AKM Technology. Concurrent Engineering (CE) Conference, Madeira, Portugal, 2003.

[MDA] Model Driven Architecture, Object Management Group, http://www.omg.org/mda/.

[UML] Unified Modeling Language: Superstructure. Object Management Group. http://www.uml.org/, 2007.

[PBL05] Pohl, K., Böckle, G.; van der Linden, F.: Software Product Line Engineering – Foundations, Principles, and Techniques. Springer, Berlin, Heidelberg, New York, 2005.

[PriDE] O/R-Mapper PriDE, http://pride.sourceforge.net.

[RJ04] Rubart, J., Jørgensen, H. D.: Collaboration and Integration based on Active Knowledge Models – Results from the EXTERNAL Project. In: Proceedings of Challenges in Collaborative Engineering, CCE'04, 2004.

[SL07] Schümmer, T., Lukosch, S.: Patterns for Computer-Mediated Interaction. John Wiley & Sons, Ltd., 2007.

[Wa04] Walsham, G.: Knowledge management systems: Action and representation. In Proceedings of the 2nd International Conference on Action in Language, Organisations and Information Systems (ALOIS-2004). Linköping University, Sweden, 2004.