Managing Knowledge on Communication and Information Flow in Global Software Projects

Kai Stapel and Kurt Schneider

Software Engineering Group, Leibniz Universität Hannover, Welfengarten 1, 30167 Hannover, Germany
E-Mail: {kai.stapel, kurt.schneider}@inf.uni-hannover.de

Abstract: Communication is a key success factor of distributed software projects. Poor communication has been identified as a main obstacle to successful collaboration. Global projects are especially endangered by information gaps between collaborating sites. Different communication styles, technical equipment, and missing awareness of each other can cause severe problems. Knowledge about actual and desired channels, paths, and modes of communication is required for improving communication in a globally distributed project. However, many project participants know little about communication and information flow in their projects.

In this contribution, we focus on knowledge about communication and information flow. It is acquired by modeling on-going and desired flows of information, including documented and non-documentated channels of information flow. We analyzed a distributed software project from the information flow perspective. Based on the findings we developed specific techniques to improve information flow in distributed software development according to the FLOW Method. In a second distributed project we evaluated one of those techniques. We found the so called FLOW Mapping technique suitable for effectively spreading knowledge about communication and information flow in global software projects.

Keywords: communication, information flow, knowledge management, global software engineering, distributed software development

1 Introduction

Communication is an essential prerequisite for the success of distributed software projects. Poor communication has been identified as a main obstacle to successful collaboration (Wolf et al. 2009). Documentation is one approach to enforce communication through a formal channel. However, documentation is a tedious task; this is one of the reasons why agile approaches suggest replacing it by direct communication. There are many other channels of communication besides documents, such as face-to-face meetings, video conferences, phone calls, text chat, or e-mail that are widely used in software development (Cockburn 2002, Stapel et al. 2009). The most effective channel, however, is face-to-face communication. It is not feasible between distributed sites in a global software project.
1.1 Knowledge on Communication and Information Flow

Management of both document-based and verbal communication is a challenge and a key competence in global software engineering. We refer to all kinds of non-documented flows of information as fluid. Document-based information flows are called solid. Global projects are especially endangered by information gaps between collaborating sites - in both solid and fluid communication. Different communication styles, technical equipment, and missing awareness of each other become severe problems (Wolf et al. 2009). Effective cooperation and collaboration in distributed projects require a good common understanding of terms, challenges, and solution approaches in the project. A proper organization of communication is a prerequisite for value-adding development activities. It is essential for distributed teams in global software development (GSD) to exchange information in a reliable, systematic way and in an appropriate form.

Improving communication in a global project requires knowledge about the ongoing and about desired channels, paths, and modes of communication. However, many project participants know little about communication and information flow in their projects (Damian, Izquierdo, Singer & Kwan 2007, Herbsleb 2007). Some process models seem to consider documents as the main containers of information in a software project. In practice, this assumption does not always hold. A lot of relevant contents are not documented (solid) but conveyed in e-mails, phone calls or personal meetings (fluid). At first glance, fluid flow may look sub-optimal, but it offers several advantages over a purely document-based process: speed, flexibility, fun, and faster feedback (Cockburn 2002). As a consequence, both documented (solid) and non-documented (fluid) information should be considered together. Effective communication in a globally distributed project will need to employ both modes of information flow.

Decision makers need to discuss issues of documentation, communication, and flow. A representation of these issues is needed as a prerequisite for effective discussions. We present the FLOW modeling approach (Schneider et al. 2008). It bridges and combines documentation and non-documented communication as two modes of information transfer. FLOW contains a graphical notation to visualize information flow in different modes and intensities. FLOW focuses on flows rather than information contents. Building FLOW models of a given project situation is a knowledge acquisition and documentation activity.

Fluid information representations must be taken seriously in order to include both verbal communication and documentation. Fluid information can be optimized together with solid information and its flow only when it is included in the models. A graphical information flow model can be used to analyze, discuss, and improve communication. We have developed different techniques to treat respective situations and information anomalies that are especially useful in distributed software development.

In this paper, we conceptualize flow models as a knowledge representation scheme about communication in software projects. It is optimized to highlight a number of aspects that are of essential importance in global software projects, including source and target of information, mode of transfer, explicit flow of experience within a project, and between projects within a software organization. We have used FLOW models for analyzing and awareness building in real software projects. Companies can benefit from FLOW models by using them as a stimulus for building support tools or by improving work practices.
Fundamental Terms

In this paper we are using terms like knowledge, information, experience, documents etc. extensively. We are aware that these concepts have overlapping meanings and different people have different definitions in mind when reading them. To avoid confusion we are specifying how we use these concepts here.

**Knowledge** is the capability and skill of a person to solve problems. Knowledge is stored in human memory. Therefore, knowledge is bound to individuals (Probst et al. 1999). In order to share knowledge it needs to be externalized as data.

“A **datum** is a putative fact regarding some difference or lack of uniformity within some context” (Floridi 2006). Data are differences in the real world, e.g. written signs, sound waves, or electric currents, that can be meaningful to an observer in a given context. Since data exists physically it can be stored and transported.

As opposed to (Nonaka 1991) we do not distinguish between tacit and explicit knowledge, but between individual knowledge (i.e. internal to the human) and physical data (i.e. external to the human), because we want to distinguish information that is explicated (e.g. in documents) rather than information that can be explicated (i.e. explicit knowledge).

A **document** is a container storing data according to a given structure.

A **medium** (i.e. communication medium, not storage medium) is a means for transmitting data.

As opposed to knowledge and data, **information** is a virtual construct without a real world correspondent. It is used as a relational concept to abstract the complex relations between the concepts knowledge and data (Stapel 2006). From the data perspective, information is data plus meaning. From the knowledge perspective, information is potential, non-personal knowledge.

**Information flow** is the conveyance of information (i.e. knowledge or data) between information storages (i.e. people or documents).

**Communication** is a special type of information flow where source and target information storages are people. Communication is a knowledge sharing activity. During communication the knowledge source, i.e. the sender, has to decide what knowledge to transmit (the message) and what knowledge to assume as given (the context) in order to successfully share knowledge with the knowledge target, i.e. the receiver. The sender then has to represent the message as data, e.g. by speaking or writing. Via a medium the message is being transmitted to the receiver. The receiver perceives the data and interprets it, i.e. associates meaning to it. Communication is only successful if the assumed context is actual context knowledge of the receiver. Sender and receiver need to have shared knowledge in order to be able to communicate successfully (Clark 1993). We call the shared knowledge on which the communication can be based **common context**. It ranges from cultural background, language skills, specific domain knowledge, and low level project details to the course of the current conversation.

**Experience** is information that stems from experience knowledge, i.e. knowledge gained by being involved (Schneider 2009). Experience information consists of a three-tuple: (1) an observation, (2) a corresponding emotion and (3) a conclusion or hypothesis drawn from 1 and 2 (Schneider 2009).

When analyzing information flow in software development we focus on project relevant information only, i.e. not on communication about personal topics.
1.2 Knowledge in the Intersection of Distributed Teams

It is not feasible to fully harmonize companies when they start to collaborate in a distributed project (Bartelt et al. 2009). There are many aspects and layers of knowledge (see Fig. 1). They build on each other: For example, company culture or agile practices have a major impact on the resulting documentation and code. Test-driven development produces different test suites and code modules than traditional development. The uniqueness and diversity of partners may be one of the reasons to get them involved in the first place. Diversity can be a strength of a distributed project (Page 2008). Full harmonization is neither feasible nor desirable. However, the opposite extreme is not feasible or desirable, either: ignorance about each other hampers collaboration.

In theory, cooperating companies could define interfaces for exchanging deliverables. Each company could then maintain its own culture, processes and communication style - as long as interfaces are being observed. In reality, however, internal practices and backgrounds will affect work products. It will be difficult or impossible to define interfaces before a distributed project starts, without adapting, refining, or changing them later. The less companies know about each other, the more surprises will they encounter, and the more adaptations and changes on many different levels will be needed during a project. We claim cooperating teams should not try to fully harmonize their processes and work practices (Bartelt et al. 2009). There are common and overlapping as well as individual and distinct aspects, as sketched in Fig. 1 (right). Each stack of aspects (left) symbolizes all the aspects that affect collaboration. They start at fundamental issues like values and company culture. Inconsistent cultures are likely to surface and clash in a situation of pressure. Terminology and specific expertise determines the position of a company in a global project. Inconsistent terminology can be at the root of misunderstandings and cause major problems. According to the process movement of the 1990ies (Paulk et al. 1994), processes affect products. Therefore, different processes and practices are important to be known explicitly - just looking at the product will not convey the whole picture. For the purpose of this paper, we focus on the upper layers of Fig. 1 (left): Knowledge of solid and fluid information flow is an essential layer that deserves explicit attention. It includes the people, solid documents, and fluid channels involved.

Fig. 1 (right) shows the stack of knowledge areas from the top: Companies A, B,
and C cooperate in a distributed project. They will need to learn about each other a little more than the pure intersection of common activities suggests. This is true on all layers of knowledge: collaborators should agree on a set of shared terms, processes, and document types, and so on. Obviously, items used by more than one partner will need to be harmonized, i.e., defined and used in the same way across companies. This intersection is the immediate interface.

Being aware of company culture, processes, and communication issues can facilitate collaboration (Desouza et al. 2006, Herbsleb & Moitra 2001). For example, it is easier to understand a specification or a piece of code if the development strategy is known: the same document would be interpreted differently in an agile vs. a waterfall environment. Increased awareness and knowledge may reduce misinterpretation and smooth cooperation. This holds on all layers.

1.3 Main Concept: Modeling Knowledge on Communication and Flow

In this paper, we suggest to focus on communication paths and media as a core structure of GSD projects. We present an easy-to-handle graphical notation for different variants and routes of information, and treat those information flow models as a specific kind of knowledge. FLOW models are complex statements comprising several intertwined facts about the communication going on in a global software project.

Along these lines, building FLOW models constitutes knowledge acquisition and documentation activities. Those models can be analyzed for suspicious elements or patterns, and desired communication situations can be modeled to compare them to real situations. We present a method to manage various activities and knowledge engineering steps in a systematic way. They contribute to eliciting, organizing, and applying knowledge and requirements about communication in software projects.

1.4 Research Method and Paper Structure

During several years of industrial experience, we have observed many situations in which communication and flow of information seemed to be disturbed. Many others have found communication to be a key to successful GSD, too (Battin et al. 2001, Bird et al. 2009, Carmel 1999, Herbsleb & Grinter 1999, Wolf et al. 2009). We decided to pursue this issue further. As Fig. 2 illustrates, a large number of vague and unstructured observations in different companies started the empirical part of our work (bottom left of pyramid, darker shade). It overlaps with the constructive branch (right pyramid, light shade). We made explicit our assumptions derived from experience. We extracted concepts to focus on the decisive aspects of communication and information flow. And, we designed support in terms of a FLOW Method to guide creation of techniques that improve knowledge acquisition and dissemination about information flows, and thus help to improve software development in general. We then conducted a pre-study to identify typical problems of distributed development using our FLOW perspective. In a follow-up constructive part we developed a specific FLOW Technique for each of the three GSD problems identified in the pre-study. To evaluate the usefulness of our solutions we selected one of the new techniques, namely the FLOW Mapping technique, and applied it in a case study (Sec. 7). Interpretation of the results and discussion of the benefits finally brings together both sides.
Since we present an approach for managing knowledge on communication and information flow, the constructive parts are at the core of this contribution. Those parts are highlighted by thick lines in Fig. 2. They are designed to support researchers and practitioners in understanding pitfalls and opportunities of information flow in Global Software Development. Evaluation and validation is explained in detail in the empirical parts. The remainder of this paper is structured according to the exchange between validation and construction. Section numbers in Fig. 2 indicate where respective aspects are treated. For example, all aspects of the pre-study are discussed in Sec. 5, while the subsequent case-study is described throughout Secs. 7.1-7.3. We frame our work by presenting an overview of related work on methods and solutions for improving knowledge management about communication in distributed software projects.

2 Related Work

2.1 Communication and Global Software Development

There is a lot of literature available about communication in the context of globally distributed software development. Especially in the past five years this field has gained more attention. In a systematic literature review da Silva et al. (da Silva et al. 2010) identified communication to be the most frequently dealt with challenge in distributed software development. The top five challenges revealed by the review are communication, cultural differences, coordination, time zone differences, and trust. In accordance to communication being a main chal-
lenge in GSD, the review found that the most solutions in the form of best practices and tools deal with communication problems as well. In a survey of 29 members of software development teams from 6 different countries Mohapatra et al. (Mohapatra et al. 2010) also identified communication mechanisms to be the most influential construct affecting coordination effectiveness in GSD.

The impact of good developer communication in GSD on software quality has been discussed, too. Wolf et al. were able to give evidence of the importance of good communication (Wolf et al. 2009). Based on analysis of social networks they were able to predict failures in the integration builds of a large highly distributed software project. Bird et al. show that it is possible to achieve comparable software quality in distributed development as in collocated software development (Bird et al. 2009). Well-known challenges of distributed development need to be overcome: Loss of communication richness, coordination breakdowns, geographic dispersion, and cultural differences (Carmel 1999). In (Bird et al. 2009), the solutions to overcome these problems were by (a) moving experienced developers from one site to the other, (b) introducing daily synchronous communication, (c) consistent use of processes and tools, (d) end-to-end artefact ownership, and (e) organizational integration.

Layman et al. (Layman et al. 2006) highlight specific needs for communication in agile projects. In a case study they identified the following four recommendations to help with eXtreme Programming based development in distributed settings: (a) define a customer authority responsible for decision making and a clear requirements statement, (b) send a key member of one team as a bridgehead to the remote site, (c) try to always respond quickly if asynchronous communication has to be used, (d) use project management tools to record and monitor project status and make the status available to everybody.

Battin et al. (Battin et al. 2001) suggest three similar solutions to overcome challenges in GSD: (a) liaisons, (b) distribute entire things for entire lifecycle, and (c) plan to accommodate time and distance. Herbsleb and Grinter (Herbsleb & Grinter 1999) offer six solutions to overcome problems associated with distance. Three are supposed to reduce the need for cross-site communication: (a) have a good design, (b) only split development on stable systems, if possible, and (c) document and publish decisions to all team members. Another three solutions address informal communication: (d) Schedule face-to-face meetings early on, (e) create a pool of liaisons who preferably are gregarious people, and (f) invest in tools that provide organizational information, maintain awareness about availability of people, and alleviate spontaneous cross-site meetings.

Another aspect frequently discussed in GSD literature that is closely related to communication is awareness. According to Gutwin and Greenberg (Gutwin & Greenberg 2002) awareness has four basic characteristics: (a) knowledge about the state of an environment bounded in time and space, (b) awareness knowledge must be maintained and kept up to date, since environments change over time, (c) maintenance of awareness is done through interaction with the environment, and (d) awareness is a byproduct of a main task to be fulfilled in the environment. In simple terms, awareness is “knowing what is going on” (Endsley 1995). A lack in awareness will lead to important communication not being carried out. For example Damian et al. (Damian, Marczak & Kwan 2007) found a significant correlation between awareness and communication frequency. In distributed settings the environment that is relevant for a task, i.e. in our case software development, is physically distributed, thus keeping awareness up to
date is more difficult than in a collocated setting. Dullemond et al. (Dullemond et al. 2010) propose that special tooling is required in a GSE setting to increase awareness. One such tool is Communico, a virtual open conversation space utilizing instant messaging for communication. They found that users of Communico felt more connected with their distributed team members, which is an indicator for increased awareness. Bruegge et al. (Bruegge et al. 2006) suggest a tool that encourages developers making communication explicit about system models, collaboration artefacts, and organizational models. They observed that usage of this tool increased awareness of relevant stakeholders and thus enables informal collaboration.

2.2 Modelling and Visualizing Communication in GSD

Laurent et al. (Laurent et al. 2010) propose a visual modelling notation for planning distributed requirements engineering projects. Similarly to our FLOW notation (see section 3.1 and 6.3 for GSD specifics) this notation has elements for visualization of sites, project participants, documents and communication flow. It also is designed to be intuitive to use and thus offers a limited set of symbols. Among the differences to the FLOW notation are its specialization on distributed requirements engineering, more visually complex symbols that demand tool support, and the ability to depict communication media, type of document and different roles of participants. Laurent et al. report that they used the models to plan and assess communication flows as well as to use them as a basis to search for communication-related problems.

Kwan et al. (Kwan et al. 2006) propose to create Requirements Centered Social Networks (RCSN) by collecting data on formal and informal flows of information. They plan to build RCSN by observing actual stakeholder communication on all available levels, including phone calls made and emails sent. They suggest collecting that information automatically and visualizing it. Computer-based communication through documents or emails is easier to capture than off-line meetings and oral communication (Damian, Marczak & Kwan 2007). Again, RCSNs were specifically designed to model information flow regarding requirements and their change. They promote awareness in a development process of changing requirements. Each diagram shows the social network of a single requirement. Nodes are individuals in a certain role, like a customer or a requirements analyst. Edges represent communication flows including informal communication. Each edge shows the date, media and frequency of a communication flow related to a single requirement.

Hansen and Kautz (Hansen & Kautz 2004) use Knowledge Maps to identify and analyze knowledge flows. The technique aims on modelling knowledge flows in order to distribute knowledge within a learning organization. Knowledge flows connect roles, individuals or organizational units, if knowledge is exchanged between these entities. These flows can be unidirectional or bidirectional. If necessary information about frequency, intensity, contents, context and importance can be attached to the flow. Hansen and Kautz identified four knowledge flow patterns: hubs, black-holes, springs and missing links. Hubs are connected to many other entities through flows; black-holes only consume knowledge while springs only produce it. Finally, missing links indicate situations where a flow should be established but it is absent. The authors successfully applied the Knowledge Map technique in a medium-size organization.

2.3 Managing Knowledge in GSD

Herbsleb and Moitra (Herbsleb & Moitra 2001) state that without effective informa-
tion- and knowledge-sharing mechanisms, the benefits of GSD cannot be exploited. E.g. team members need current information about market needs or location of expertise in order to exploit new opportunities, the benefits of a diverse team, and reuse existing solutions. In the same line Desouza et al. (Desouza et al. 2006) conclude from the analysis of knowledge management practices in more than 50 software organizations that managing knowledge about important information and who possesses it is essential to avoid duplication of work at different sites. From analysis of two case studies Boden et al. (Boden et al. 2009) found that especially in small software companies managing knowledge about who knows what, who is working with whom, and who is working on what is more important than following a standardized procedure.

These studies – like many others – emphasize the importance of communication and the management of information flow in distributed software development. However, one important aspect of communication is often ignored or only observed on a very high level of abstraction: verbal, synchronous, informal, and ad-hoc communication. We assume this may be due to the difficulties associated with capturing, visualizing, and managing this kind of information flows consistently with other communication relationships. Our concepts to overcome the difficulties in capturing, analyzing, and improving informal communication and incorporating them consistently with analysis of planned communication are being developed in project FLOW.

3 FLOW Fundamental Concepts

In this section the basic concepts of information flow analysis are presented. The FLOW project was initiated at Leibniz Universität Hannover in 2004. Since then we investigate characteristics, modeling, analysis, and improvement of information flows in software development. Knowledge about the sources and flows of important information helps to better understand and improve software development projects.

The approach of information flow analysis is based on some fundamental beliefs and assumptions that stem from a combination of established Software Engineering knowledge and our experience in software development in industrial and educational settings:

- The most important resource in software development is information. Whether it is product information, which is all information that is needed so that the final software product functions as desired, like requirements and design decisions, or control information, which is all information that is needed to create the software, like programming skills and best practices. A lot of information resides as knowledge in peoples’ minds – as opposed to be written down in documents. This viewpoint highlights the importance of people in software development (Cockburn 2002, Davis 1995).
- Success of a software project highly depends on the adequate flow of information. The correct requirements need to be identified and channeled through the development phases until they are accurately implemented in the final software. Experiences from former projects, like what development method worked well and what didn’t, need to be routed to the right place in the current project in order to avoid repetition of mistakes and repeat successes.
- Modeling and analyzing information flows visually is the base for understanding and improvement (Moody 2009).
- Coarse modeling of information content is sufficient (Ambler 2002). Often, only abstract levels of information are modeled on their way through the project, like requirements or experiences. In spe-
cial cases more concrete types of information may be appropriate.

- **Experience is an especially valuable kind of information** in software development which is being modeled explicitly. It often positively influences activities and acts as a catalyst (Basili et al. 1994).

- We introduce – as a key concept – the state of information:

<table>
<thead>
<tr>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Solid information</strong> is information that is (1) long term, repeatable accessible and (2) comprehensible by others in a given scope.</td>
</tr>
<tr>
<td><strong>Fluid information</strong> is all information that is not solid.</td>
</tr>
</tbody>
</table>

The scope determines who the “others” are and how long “long term” is. For example project documentation for later maintenance usually needs to be more exhaustive than documentation that is only used within the project to qualify as solid information, because the target audience of the former has less project context knowledge than the audience of the latter and thus can only comprehend the information if enough context is being provided within the documentation. Also, a whiteboard drawing of the system architecture can be considered solid in a short project scope when everybody involved still remembers its design rationale and nobody clears the whiteboard, but in a long lasting project people eventually forget design decisions or someone wipes the board so the information on the board is lost. Typical examples of fluid information are verbal or not objectively reproducible information like informal e-mails and personal notes because they are either not long term accessible or not comprehensible by others since important context information cannot be retrieved anymore (e.g. the context of an e-mail conversation).

- Finally, the information flow perspective allows to conjointly analyzing and improving conventional projects, which are based on document centric processes, as well as agile projects, and thus gain the benefits of both worlds.

Two of these points require further explanation. Referring to information metaphorically as either fluid or solid states of information is typical for FLOW. It points out that the same information can appear in different shapes. Different states implicate different characteristics similar to fluid and solid matter.

- Solid information can be recalled at any time. It is stored in documents or recordings (video or audio). Access as well as storage cost time and effort, but they are repeatable. Solid experience is a special kind of solid information. It is available through checklists, best practices and patterns, such as design patterns.

- Fluid information, on the other side, is bound to people or other volatile media. We call fluid whatever someone has in mind and which cannot be obtained or reproduced by third parties. Fluid information can easily be transmitted by conversations and supported by some handwritings. But fluid information can easily be lost or forgotten. Fluid experience slowly grows in a person’s mind while the person is doing or observing something.

Fluid information flows like face-to-face conversations are often very efficient. Only small pieces of information need to be sent. Receivers have a lot of pre-knowledge which helps them to correctly put the pieces in context and understand the conveyed meaning. Face-to-face conversations usually build on a large body of common knowledge (Clark 1993).
DEFINITION

Project context knowledge is all prior knowledge on all aspects of a project that a person needs in order to correctly interpret relevant information for that project.

Project context knowledge has many levels. It ranges from cultural background and language to specific domain knowledge or low level project details (see Fig. 1, left). Context knowledge is usually not shared extensively during a professional conversation. Context knowledge is rather assumed to already be in place before a conversation starts. This assumption may hold for collocated teams where people have worked together for a long time and came to know each other. In globally distributed teams members come from different cultural, educational, and professional backgrounds. Hence, only very limited common context knowledge can be assumed. This lack of common context knowledge places a significant burden on communication in distributed projects.

According to the above definition every developer in a software project needs project context knowledge to reduce broken information flows and misinterpretations. We believe that the ideal amount of shared knowledge between distributed sites is somewhere between knowing only the obvious and knowing everything about the others as sketched in Fig. 1 (right). It is not required to fully harmonize all aspects of all collaborating partners. However, ideal overlap exceeds the minimal intersection of commonly used artifacts, processes etc. It is good to know a little more about the background of partners, even if this knowledge is not directly required to carry out any given task. This is also true for knowledge about communication and information flow.

Solid and fluid states and flows of information are the core concepts of FLOW. Explicitly modeling experience and the notion of context knowledge set the scene for analyzing project situations from an information flow perspective. This is what we call the FLOW perspective. A specialized notation is needed to highlight the just-mentioned main concepts of information flows in a FLOW model. This notation should be graphical so that humans can easily create and work with FLOW models.

3.1 The Graphical FLOW Notation

Information flows could be represented as data flow diagrams (DFDs) as introduced in the 1970s (DeMarco 1979). The basic idea was – and still is – not to follow the control flow like in most process representations. It is rather important where information flows. When putting information flows in the center of attention it is more important to see who knows what rather than seeing who does what. A visual notation helps to clarify certain information flow situations and patterns. It is supposed to transport the basic concepts and at the same time be open for interpretations. After all, it is a medium of communication for people and not a programming language. The graphical notation for FLOW models is supposed to comply with the following requirements:

- Has to be easy to understand. Use only a few, intuitive symbols.
- Distinguish and visualize fluid and solid information storages and information flows.
- Express experience as a special type of information.
- Means of expressing that information is not flowing.
- The ability to establish relations to existing process notations.
- The ability to handle complexity by building hierarchically connected models.
- The ability to handle abstraction by modeling flows and storages which do not have a state yet.
Some of these requirements, for example to visually distinguish between solid and fluid, led to the creation of a new notation instead of re-using DFDs. Very simple but flexible notations were most appropriate wherever we used information flow analysis so far (Schneider 2004, Schneider & Lübke 2005, Schneider et al. 2008, Stapel et al. 2008, Stapel et al. 2009, Stapel et al. 2007). Table 1 summarizes the symbols that satisfy the mentioned requirements and form the graphical FLOW notation.

The document symbol represents a solid information storage. A document is the most prominent representative of solid information storages in software development, which is the rationale for choosing that symbol. A set of two or more documents (of the same type) is depicted by a triple document symbol. Information flows originating from a solid storage are themselves solid. They are depicted by an arrow with a solid line. A smiley symbol represents a fluid information storage. It was chosen since fluid information is mostly bound to people. A group of two or more persons is depicted by a triple smiley. If it is necessary to distinguish between roles and individuals, different caption styles can be used, i.e. an underlined caption for roles and a regular one for individuals (for example: "Johnson" as opposed to "Analyst").

Information flows originating from a fluid storage are fluid, too. They are depicted by an arrow with a dashed line. A combination of a document and a smiley symbol represents an undefined information storage. Undefined storages are used whenever the state of the contained information is unknown or not yet defined when building a FLOW model. Information flows originating from an undefined storage are undefined by definition. They are depicted by an arrow with a dash-dotted line. Information flows originating from solid storages are solid: It depends on the originating storage whether the flowing information can be reproduced and understood by others.

In software development experience is an extraordinary valuable type of information. Therefore, the FLOW notation has an explicit means for depicting experience storages and flows. Usually a different color is used. Gray should be used for black and white printouts. Sometimes it might be important to specifically mark a not existing flow between two storages. This can be achieved by the null flow symbol, an orthogonal line at the end of a flow.

The activity symbol has two functions in a FLOW model:

- **Hierarchization:** An activity is a collection of information flows and other activities and thus can summarize and hide them. Hence, it facilitates hierarchically structured FLOW models. Incoming and outgoing flows, the so called FLOW interface (Schneider & Lübke 2005), have to be consistent with the underlying detailed model.

- **Process link:** It is used to connect FLOW models to process notations. By this, aspects of information flows can be integrated in existing process models. Incoming and outgoing information flows can be attached to the activity symbol from different sides to depict the difference between information flows that are involved in an activity with regards to content and information flows that are controlling the activity. Information that controls an activity is connected on top or bottom of the activity symbol. In software development information that controls an activity often stems from experience. Information that is being or was altered and transformed in the activity is connected from left or right.
4 The FLOW Method

Throughout this paper, we use several terms on FLOW that are highly related to each other:

**DEFINITION**

**FLOW is an approach** of explicitly capturing, modeling, and using flows of information. It considers both solid and fluid information, and regards experience as a special kind of information (Schneider 2009).

A **FLOW model** is a graphical model of solid and fluid information storages, flows, and experiences.

The **FLOW Method** is a theoretical framework for applying the above-mentioned concepts of FLOW to improving software development. It contains the **FLOW improvement process** which consists of a preparation phase and three improvement phases (see Fig. 3).

A **FLOW Technique** describes concrete activities to be taken to reach a certain improvement goal. The practical application of the FLOW Method is done through FLOW Techniques.

Fig. 3 shows the FLOW improvement process including the preparation phase and the following iterative improvement cycle. The goal of an information flow improvement effort is determined in the preparation phase. Additionally, the specific techniques to reach that goal are chosen. An improvement cycle consists of the three phases (1) capture, (2) analyze, and (3) improve. Deming’s Plan-Do-Check-Act cycle (PDCA) follows a similar pattern (Deming 1986). Basili’s Quality Improvement Paradigm (QIP) (Basili et al. 1994) is even more sophisticated and contains more steps for experience handling. We decided to build on a simple denominator consisting of only three phases. We aimed at a structure for information flow based process improvement that is generic enough to fulfill a wide variety of goals (see FLOW Goal in 4.1 Preparation), and contains the essential tasks of information flow analysis. Each particular technique can instantiate our simple three-phase scheme. QIP and PDCA are both designed for organization-wide improvement endeavors whereas the FLOW Improvement Process can also be applied to single projects or even single development activities. In fact, the FLOW Improvement Process can be incorporated within a QIP or PDCA cycle, e.g. as the execution phase in QIP.

The phases of the improvement process are at the core of the FLOW Method. Each phase has an outcome that is needed for the following phase. For example, the capture phase provides a FLOW model that is needed in the analysis phase and the results of an analysis are needed for improvement. The phases capture, analyze, and improve can be performed iteratively during process improvement. Thereby, the effects of an improvement can be captured and analyzed in a follow-up iteration. It depends on the respective FLOW Technique how information flow is captured, analyzed, and improved in detail. A FLOW Technique describes activities for one or more phases of the improvement cycle.

![Fig. 3: The iterative FLOW Improvement Process](image-url)
Table 1: Symbols of the Graphical FLOW Notation

<table>
<thead>
<tr>
<th>State of Information</th>
<th>Storage</th>
<th>Flow</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solid</td>
<td>1</td>
<td>2..n</td>
<td>Control*</td>
</tr>
<tr>
<td></td>
<td>&amp;&lt;br&gt;</td>
<td>&amp;&lt;br&gt;</td>
<td>&amp;&lt;br&gt;</td>
</tr>
</tbody>
</table>
|                      | <Document><Documents><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Content><Contents
Table 2: Example FLOW Goals

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Scope</th>
<th>Time</th>
<th>Concrete FLOW Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understand</td>
<td>activity</td>
<td>ongoing</td>
<td>Understand requirements elicitation during an interview.</td>
</tr>
<tr>
<td>Understand</td>
<td>activity</td>
<td>afterwards</td>
<td>Understand a review session through a post-mortem analysis.</td>
</tr>
<tr>
<td>Improve</td>
<td>activity</td>
<td>beforehand</td>
<td>Tailor specification review activity beforehand.</td>
</tr>
<tr>
<td>Improve</td>
<td>project</td>
<td>ongoing</td>
<td>Improve requirements flow during a project.</td>
</tr>
</tbody>
</table>

- **Purpose:** The purpose of a FLOW Goal can be to either capture, understand, or improve information flows. Usually, a FLOW technique subsequently serves all three purposes: First, the information flows need to be captured. After that they can be analyzed and understood. Finally, the flows can be improved.

- **Scope:** The scope of a FLOW Goal can be activity, project, or organization wide. On the activity level the information flows of a single software development activity are considered, e.g. the requirements elicitation activity. On the project level flows of a whole development project are integrated. Whereas, on the organizational level cross project information flows are at the center of attention.

- **Time:** The main activities of a FLOW Technique are either executed before, during, or after a project. Hence, a FLOW Goal has a time aspect that can take one of the three values beforehand, ongoing, or afterwards.

Theoretically, there are 27 different permutations of these three aspects. But some combinations are not meaningful, such as improvement efforts to be undertaken after the life-time of an organization. Examples of typical meaningful FLOW Goals are shown in Table 2.

2. **Collect project parameters.** Knowing the project parameters is also important in order to choose the right FLOW technique. Relevant parameters are:

- **Project size:** Project size can be given as team size and/or as budget size. Both team size and budget size are important to determine how complex the FLOW models are likely to be and how many resources can be spent to prepare and execute the FLOW Techniques.

- **Domain:** The problem domain of the product, e.g. automobile or finance, and the domain of the desired technology, e.g. web or embedded, may influence the choice of an adequate FLOW Technique, and thus are important to know upfront.

- **Development style:** The development style, e.g. agile or process driven, has a high influence on the choice of a technique since frequency and type of information flows highly depend on the style of development.

- **Distribution:** Communication in projects where the team is collocated is different from communication in distributed projects. Some FLOW Techniques are specifically designed for a distributed setting. Also, the type of distribution might be important to know in order to choose a FLOW Technique, since information flows tend to be different if the team is distributed along process phases, e.g. design and implementation, or within a phase, e.g. distributed sub-teams working on the same source-code.

- **Other constraints:** All other constraints, which could be important in the choice of the right FLOW Technique, should also be noted.
### Table 3: Classification template for FLOW Techniques

<table>
<thead>
<tr>
<th>Name of technique</th>
<th>__________________________________________________________________________</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase</td>
<td>☐ Capture ☐ Analyze ☐ Improve</td>
</tr>
<tr>
<td>Goal</td>
<td>☐ Capture ☐ Understand ☐ Improve</td>
</tr>
<tr>
<td>Purpose</td>
<td>☐ Activity ☐ Project ☐ Organization</td>
</tr>
<tr>
<td>Scope</td>
<td>☐ Beforehand ☐ Ongoing ☐ Afterwards</td>
</tr>
<tr>
<td>Time</td>
<td>☐ Beforehand ☐ Ongoing ☐ Afterwards</td>
</tr>
<tr>
<td>Designed for project parameters</td>
<td>__________________________________________________________________________</td>
</tr>
<tr>
<td>Project size</td>
<td>☐ Team size: _____________________ ☐ Budget: _____________________</td>
</tr>
<tr>
<td></td>
<td>☐ Resources needed: _____________________________________________________</td>
</tr>
<tr>
<td>Domain</td>
<td>☐ Problem domain: ______________________________________________________</td>
</tr>
<tr>
<td></td>
<td>☐ Technical domain: _____________________________________________________</td>
</tr>
<tr>
<td>Development style</td>
<td>☐ Agile ☐ Process ☐ Other: _____________________</td>
</tr>
<tr>
<td>Distribution style</td>
<td>☐ Local ☐ Distributed (Type: _____________________)</td>
</tr>
<tr>
<td>Other</td>
<td>________________________________________________________________________</td>
</tr>
</tbody>
</table>

#### 3. Choose techniques. The final step in the preparation phase is to find FLOW Techniques that best fit the goal and properties that were determined in the previous two steps. To facilitate this, each FLOW Technique is classified by an instance of the template given in Table 3. Each technique has a descriptive name. It provides concrete activities and supporting tools for one or more of the FLOW improvement process phases capture, analyze, and improve. A technique provides support for different aspects of a FLOW goal. These aspects are marked in the goal section. Multiple selections are possible since some techniques can be used to reach different goals, e.g. to improve a single activity and to improve a whole project. If a technique was specifically designed for certain types of projects this can be noted in the project parameters section. For example, if a technique was developed for agile projects only, it would be noted in this section.

FLOW techniques are classified using the template in Table 3 and listed. For any given improvement task, suitable techniques can be selected by matching the results of the preparation phase with the classified techniques.

In order to apply FLOW based process improvement a project or an organization needs to be willing to be observed and open for improvement suggestions. Typically, the FLOW Method is being applied by an external FLOW expert, by internal process improvement staff, or by a project manager. Another prerequisite is that sufficient FLOW Techniques are available that match the FLOW goal. If these preconditions are met and preparation is finished the FLOW improvement cycle can be started.
4.2 The FLOW Improvement Cycle: Capture, Analysis, and Improvement

A FLOW Technique supports at least one phase of the FLOW improvement cycle. The consecutive phases of the improvement cycle are capture, analyze, and improve. Each phase produces a prerequisite for the following phase. A FLOW model is the output of the capture phase. This FLOW model is the basis for analysis in the analysis phase. Typically, the model is being visualized with the FLOW notation so a FLOW expert can analyze it visually. The results of the analysis are used to plan and execute improvements. The improved information flows can be the start of a new iteration of the cycle. As opposed to the solid state of the captured FLOW model, whether the findings of an analysis or the improved information flows are documented or not is not specified by the FLOW Method, it depends on the concrete FLOW technique. The FLOW model in Fig. 5 summarizes the activities and information flows of the FLOW improvement cycle.

Concrete techniques are needed to implement the FLOW method. These techniques prescribe activities for each of the phases of the improvement cycle. To test the applicability of the FLOW Method in distributed setups we conducted a pre-study (Stapel et al. 2009) to identify information flow related problems in distributed projects (see section 5), developed specific FLOW Techniques to conquer these problems (see section 6), and evaluated one of the techniques in a second case study (see section 7).

5 Pre-Study Minotaurus: Identify Problems

Applying information flow concepts as provided by FLOW can help to identify communication challenges in global software projects. We participated in the Minotaurus software project which was distributed over three locations and three time zones to identify problems of distributed development using the FLOW perspective. Teams in Finland, Russia, and Germany participated in the project. This section provides a summary of the setup, our analysis approach and the three information flow related problems we identified.

5.1 Pre-Study Project Setup

Project Minotaurus was a globally distributed software development project joining a total of 10 developers from Finland, Russia, and Germany. Each site had 3 developers involved. A 4th Finnish developer joined the Russian sub-team as an ambassador. All developers were graduated students with at least 3 years experience in software development. Developers from Russia and Finland already knew each other from previous projects. The German developers were new to the distributed team. They joined the global team for two weeks. The development goal of the Minotaurus project was the extension of the Trac\(^1\) open source project management tool. It had to be extended by special features for agile development. The programming language was Python. Requirements were derived from two earlier

\(^1\) http://trac.edgewall.org/
versions of Minotaurus, and new requirements were added by the research group at VTT, Finland. A SCRUM-based process was used for project management and coordination. SCRUM Master and Product Owner were located in Finland. The project lead was located in Finland as well. The backlog was maintained in a Wiki. Trac itself was used for ticket management, and Subversion was used for version control. The following media were used for communication: Wiki, text chat, phone, video, and desktop sharing. Video and desktop sharing was used rather seldom due to technical difficulties.

The total development time of the project was 5 weeks. Each iteration (or "sprint" in SCRUM terminology (Schwaber & Beedle 2002)) took one week. Our results and conclusions were mostly based on the data we gathered during the two weeks of German participation. A video summarizing selected communication aspects of the project is available at www.se.uni-hannover.de/en/glose/minotaurus.

5.2 Observation and Analysis Approach

Our intention was to study information flow problems in GSD. Developers usually directly experience such problems during project runtime. Hence, we used observation packages for supporting elicitation and analysis of experiences (Schneider 2009). An observation package is a one-page form designed as a light-weight tool for raising and capturing authentic and useful experiences during an ongoing project.

**DEFINITION**

According to Schneider (Schneider 2009), each piece of experience consists of
- an observation (what did I observe?)
- an emotional reaction (what did I like or dislike about it? Why did I care to write it down?) and
- a conclusion (what was the reason or consequence of the observed situation?)

Please note that this definition targets experience-based improvement. Therefore, it highlights key aspects of an experience. Observation packages are used as a low-threshold paper tool to capture experiences. They need to be compared, combined, and rephrased during experience engineering (Basili et al. 1994, Basili & McGarry 1997, Schneider 2009). Our goal in this pre-study was to better understand problems related to information flow in global software development. Therefore, we enriched generic observation packages to include distribution-specific check boxes for classification (right part of Fig. 6). The labels of the checkboxes were chosen so that the cognitive effort of the developers who had to fill them in wouldn’t be too high, but still provide us with the possibility for presorting the results. This leads to a classification scheme that developers can quickly grasp but is neither mutual exclusive nor exhaustive.

During the two weeks of the German participation in the Minotaurus project the developers completed 44 observation packages. According to the specific categories for GSD experiences, we found the distribution of perceived problems shown in Fig. 7.

Analysis showed that most of the experienced problems were related to communication. This finding was not surprising in the context of distributed development (Bird et al. 2009, Carmel 1999, Curtis et al. 1988, Grinter et al. 1999). Follow-up analysis of recorded communication and of the observation packages is illustrated in Table 4, 5, and 6. We were able to identify three reasons for the communication problems:

1. Missing context knowledge for interpretation of information (see 5.3).
2. Missing documented information (see 5.4).
3. Missing awareness (see 5.5).
We briefly describe these three reasons with respect to some of the observation packages. The brief discussion is supposed to provide an impression of the FLOW perspective and its impact on looking at communication issues.

### 5.3 Missing Common Project Context Knowledge

We extracted the observation and emotion parts from observation packages. Audio and video recordings were available to substantiate claims or concerns. The following observations provide some concrete feedback.

**Table 4: Obs. related to missing context**

<table>
<thead>
<tr>
<th>ID</th>
<th>Observation and emotion part of experience</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>After a meeting a developer was angry about the wasted time, because he didn’t know what it was they just talked about, how it works and where to find it in SVN.</td>
</tr>
<tr>
<td>14</td>
<td>Finnish and Russian developers talk about last iteration. The Germans cannot follow. They are afraid they missed important information.</td>
</tr>
</tbody>
</table>

The (ID 6) problem is obviously related to missing context knowledge. Developers do not have sufficient knowledge about the project status, goals, and implementation. SVN is considered a solid store of information. The other developers had no problem following as they had received the information in fluid form (did not need SVN). It is embedded in their prior knowledge and, therefore, accessible for understanding. (ID 14) deals with a very similar phenomenon. Germans entered the project later than the others; it is natural they would not have the same amount of prior context knowledge. It is important to note the emotional responses in both situations: One group is afraid they missed important information - but is not sure they did, or which information they may have missed. In (ID 6), the developer even gets angry. In both cases, seamless collaboration is not feasible. There will even be emotional barriers to overcome before all groups are able to work together properly. In terms of Fig. 1 the overlapping area of shared context knowledge was too small. Two communication partners had a larger overlap and could communicate easily, while the third partner was not able to follow - and will likely not be able to continue effectively and efficiently.
5.4 Missing Documented Project Information

In the previous block of observations, knowledge on conversation partners or roles was missing. In addition, content was lost or became inaccessible. Important knowledge was not documented and could not be found when it was needed: Such knowledge included requirements raised, decisions made, and rationale associated with those decisions. Several of the observed problems clustered around fluid information that was forgotten or lost. Missing information caused significant confusion and frustration among developers.

Table 5: Observations about missing documented information

<table>
<thead>
<tr>
<th>ID</th>
<th>Observation and emotion part of experience</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>Russian developers were searching for some requirement that had been mentioned in a distributed meeting. They could not find it on Subversion.</td>
</tr>
<tr>
<td>10</td>
<td>Trac tickets were out of date. Documented and discussed status diverged.</td>
</tr>
</tbody>
</table>

Documentation takes a lot of time and effort both for writing and reading, which may slow down development. Developers are often averse to documentation. Managerial and control information is important to solidify and store in a common repository, but requirements and technical details also need to be solidified. Nevertheless, agile approaches encourage direct communication. Documentation is reduced on purpose. For example, requirements are not necessarily documented in a shared repository. A subteam may simply implement required features; the requirement is embedded in the code or test code. Solid and fluid information do coexist in almost all projects, but when documented information is inconsistent with fluid information status, this constitutes an information flow problem. As a consequence, developers are not sure what they can use as reference - and whether there may be even newer information.

5.5 Missing Group Awareness

It makes a big difference what developers know about each other, and about remote sites (see Fig. 1). Developers cannot consider issues they are not aware of. We found that many developers are not aware of what is going on at the other project sites. In our observations, they did not know who was supposed to decide, and they were worried about that uncertainty.

This is true for knowledge about communication and information flow. If a GSD project is supposed to exchange information effectively, good awareness of communication and information flow issues is required. There may be problems with group awareness in collocated teams, but distributed teams seem to have much more difficulties perceiving the entire distributed group of developers as one team. Some of our observations emphasized this very strongly:

Table 6: Observations related to awareness

<table>
<thead>
<tr>
<th>ID</th>
<th>Observation and emotion part of experience</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>Developers discuss about a requirement. It is unclear who decides what to do. They were worried whom to ask when uncertain.</td>
</tr>
<tr>
<td>39</td>
<td>After a Skype conversation two developers reported that they were not sure who had been their conversation counterparts. They felt very confused about that.</td>
</tr>
<tr>
<td>44</td>
<td>Towards the end of the project, developers reported that they had only short audio Skype meetings with the other sites; they did not even make an attempt to identify the people on the other ends.</td>
</tr>
</tbody>
</table>

In (ID 29), developers were not aware of an essential project role: since they did not know who could provide requirements, they were worried or uncertain. Effective communication cannot occur if the appropriate partner is not known. In collocated teams,
this kind of awareness grows with many interactions. It is often fluid.

(ID 39) represents a different problem: Not everyone knows everybody else by name, face, or voice. After a while, no one dares to ask who is on the line - so an entire conversation can be carried out without realizing who was talking to whom. Awareness of team members across distributed sites turned out to be very limited. This is remarkable, since Minotaurus was a relatively small and short project. Even there, the team had not developed an awareness that exceeded single sites. In (ID 44), this effect has happened several times. Developers felt confused at first (ID 39), but do not even care after some repetitions. As a consequence, conversations do not add to a sense of familiarity or a better awareness. Unless a certain threshold has been overcome, on-going communication increases frustration and confusion rather than common knowledge.

Our local observations showed a number of deficiencies in detail: Many times the team was not even aware of who was currently present in the development room at the two other locations. Often, the tasks of these people were not known. People do not know enough about other sites, so they cannot develop useful expectations about them. Developers are not able to conduct their work since they have no good mental model of their distributed environments. Competent behavior requires an appropriate mental model. Developers do not even know that they do not know some things.

Real group awareness can emerge only when developers know who constitutes the group, who is working on what part of the project, and who may be reached at a given point in time. This issue was addressed by a FLOW technique, which is described below.

6 FLOW Techniques: Improve GSD from FLOW Perspective

In this section we bring together the FLOW fundamentals, the FLOW method and the problems identified in the pre-study to build three FLOW techniques that are specifically designed for global software development: (1) The context sharing ambassador to conquer the problem of missing project context, (2) the solidification as a by-product technique to conquer the problem of missing documentation, and (3) the FLOW Mapping technique to conquer the problem of missing team awareness. Table 7 summarizes the classification of all three techniques according the classification scheme presented in section 4.1.

| Table 7: Classification summary of presented FLOW techniques with application criteria; ";-;" means "no constraint, works for all" |
|--------------|-----------------------------|-----------------------------|-----------------------------|
| Technique    | Context sharing ambassador | Solidification as a by-product | FLOW Mapping               |
| Phase        | Analyze and improve         | Improve                     | Capture, analyze, and improve |
| Goal         | Purpose                     | Improve                     | Understand or improve       |
| Scope        | Project                     | Improve                     | Activity or project         |
| Time         | Ongoing                     | Activity                    | Ongoing                     |
| Designed for … |                             |                             |                             |
| Project size | More than 8 developers      | -                           | 4 to 40 developers          |
| Domain       | -                           | -                           | -                           |
| Development style | -                      | -                           | -                           |
| Distribution style | Distributed     | -                           | Distributed (horizontal)    |
6.1 Context Sharing Ambassador

The task of the context sharing ambassador is to increase the shared knowledge between sites (see Fig. 1 right: extending the gray area). We claim that this can be accomplished by sending ambassadors to remote sites who especially act as context sharing catalysts. Identifying important project context is difficult. But an ambassador who regularly attends meetings or even problem negotiations will soon learn about the foreign sites’ context on many levels (see Fig. 1 left). During the meetings the ambassador should explicitly pay attention to the following differences between the home and foreign team:

- Educational background
- Professional experience
- Process knowledge and experiences
- Domain knowledge
- Knowledge about tools
- Former co-workers
- Roles in project, department, company
- Legacy systems
- Work environments
- Habits, conventions and corporate culture
- Language barriers
- Rationale behind decisions
- Well established sources of information like Intranets, Wikis, or other experience factories.

This list is an experience-based checklist of issues that we found useful. It should be used like a checklist by the context sharing ambassador. When new issues arise, they can be added to the list. This mode of continuous rework and improvement is typical for experiential learning approaches (Bomarius et al. 1998, Brown & Duguid 1991, Kolb 1984).

The so acquired knowledge needs to be reported back to the home site. This can be done either directly, e.g. by calling, or through a dedicated document which could be reused in future projects joining these sites. Fig. 8 illustrates the information flows and activities of the context sharing ambassador technique.

Identifying and sharing project context between the distributed teams makes project communication more effective because misunderstandings are less likely and communication can be based on a greater source of common knowledge. Improving communication will improve the ongoing project, so the FLOW goal of this technique is reached.
6.2 Solidification as a By-Product

In distributed software projects different types of communication media are used for project communication to substitute the missing possibility of face-to-face meetings. Using these communication media for (semi-)automatic documentation can save a lot of effort. Fig. 9 shows the idea of the solidification as a by-product technique. Documentation of information from meetings and corresponding results can be created automatically by using an especially designed by-product tool during the meeting that takes place between developers usually located at different sites. This documentation can be reused later by the developers. The by-product tool has to be custom-built for the development activity it is used for and for the communication media that is being used during execution of the activity. For example, documenting results from a daily stand-up meeting that is being held via phone conference imposes other requirements on a by-product tool than a collocated software prototype demonstration (Schneider 1996). Therefore, a by-product tool is always specific for a combination of an activity and a communication medium. The creation of a particular by-product tool is not part of the FLOW technique although it is a prerequisite.

Solidifying information as a by-product helps to save important knowledge for reuse with just a slightly increased effort. By this the solidification as a by-product technique improves a dedicated software development activity while it is being executed and thus reaches its FLOW Goal.

6.3 FLOW Mapping

The goals of the FLOW Mapping technique are:
- to increase team awareness in distributed teams and
- to initiate well-directed point-to-point communication between the developers at different sites.

To achieve these goals the technique is centered around the visualization of a FLOW Map. A FLOW Map is a special FLOW model extended by the following features:
- Assignment of stores to locations, e.g. development sites
- Line width emphasizes strength of information flow
- Undirected connections represent information flows in both directions
- Additional meta-information about each store, the so called yellow pages: contact information, picture, local time, status information, role in project, skills relevant for project, current task or work item, etc.

A FLOW Map visualizes project participants as fluid stores, their project locations, their yellow pages information, and their communications among each other. An example FLOW Map from our second case study (see next section) is illustrated in Fig. 12.
In order to reach the goals of the FLOW Mapping technique a number of specific activities need to be carried out. These activities are clustered according to the FLOW Method. Fig. 10 shows the activities and information flows of the FLOW Mapping technique for the phase capture whereas the activities of the phases analyze and improve are shown in Fig. 11.

We will describe the key activities of the capture phase in detail:

**Create initial FLOW Map:** Before starting a distributed project the teams have to be planned. This can be done based on the creation of a FLOW Map. Thus, the FLOW Map becomes a utility for project planning (Stapel et al. 2011). First, the coordinator of the project needs to establish the team. She needs to decide what developer from what site will be part of the global project team. Additionally, the coordinator needs to plan the communication channels to be used during the project. Will e-mail be sufficient for requirements elicitation? How many collocated meetings are needed and can they be afforded? Do we have the technology to use high definition video conferencing for daily stand ups? These are questions that are part of the activity plan communication channels. The last prerequisite for the creation of the initial FLOW Map is the contact information that each developer needs to supply for the yellow pages in the FLOW Map. Based on the established team, the planned communication channels and the contact information of each developer the initial FLOW Map can be created by performing the following 6 steps:

1. Draw dedicated areas for each participating site.
2. Draw a fluid store for each team member located at the respective site.
3. Draw solid stores for all documents or data stores that are essential to exchange project information.
4. Sketch a fluid flow between two people if they are supposed to exchange information on a regular basis during the project. Put an arrow in if the information will mainly flow in one direction rather than the other. Sketch a fluid flow between a person and a document if the person is supposed to regularly solidify information in that document. Use different line widths to emphasize frequent or intense flows. The richness of the intended communication channel also influences the line width.
5. Sketch a solid flow between a document and a person if the document is supposed to be read regularly during the project.
6. Finally, the FLOW Map will be complemented by at least the following yellow pages information for each participant:
   - Portrait images of the developers
   - Contact information like e-mail address, phone number, or Skype ID, corresponding to the planned communication channels
   - Role in project
   - Skills relevant for the project
   - Current tasks or work items

The initial FLOW Map can now be used during the project to show the developers who else is participating at remote sites and how these teammates can be reached for communication during the project. Furthermore, it shows intensity of the communication as planned. This gives the developers a hint on who they should regularly communicate with. Usually, project settings are not static. Roles may change. Tasks definitely change throughout a project. Desired communication paths may also change. Therefore, only an up-to-date FLOW Map can increase team awareness continuously. Another increase in awareness can be reached by showing current communication activities instead of desired information flows. This also is only possible if communication events are observed and timely updated on the map.

**Update FLOW Map:** Only an up-to-date FLOW Map will reach its full potential, since we assume that developers will lose interest in using the map if it contains outdated information. Important changes that need to be updated as soon as possible are:

- Any change in the yellow pages information of the developers. That includes changes in roles, tasks, work items, or constitution of sub-groups like pairs, if applicable
- Changes in desired communication behavior, or even more challenging
- Current communication events.

**Communication Monitoring:** To facilitate the latter, communication events have to be monitored and visualized in the map. A communication monitor can either be a person who observes the project or a tool that automatically logs communication events. Communication over digital media can be monitored automatically more easily than face-to-face communication. All non-automatically collectable events have to be supplied manually either by the developers or the coordinator in order to be timely updated in the map.

During the project (see Fig. 11) the coordinator and the developers are analyzing project communication by visually observing the ideally constantly updated FLOW Map. They can see if remote colleagues are

---

**Fig. 11:** Activities of the FLOW Mapping technique for the phases analyze and improve
busy in a meeting or communicating through other channels, who is working on what task, and who is expert in what field. Along with the contact information this knowledge can be used to initiate point-to-point communication in case of a problem or question. Also, the coordinator can better control communication, for example by planning and starting group meetings or by bringing together two developers who are working on the same problem.

**Compare actual to initial FLOW Map:**
During the project the current up-to-date FLOW Map can be compared to the initial FLOW Map in order to detect deviations from communication as planned. If people do not communicate as planned, or do not send documents as expected, there is a chance information will not get where it is needed in time. In some cases, traditional communication practices lead to different paths than the ones designed by the project leader. This may or may not be a problem, but should be known for future planning. In some cases, project participants may be more creative and use more and richer channels of communication than initially expected. In all cases, learning more about the actual communication practices will increase the project context knowledge. Everyone who uses a FLOW Map will gain knowledge and awareness about the information flows in the project.

In summary, visualizing project participants and their communication paths using the graphical FLOW notation improves knowledge about each other among the distributed team members. Developers in distributed teams now know who is participating, who is expert in what field, and who is working on what. All are prerequisites for a well-directed point-to-point communication in case a developer has a question: she needs to know who might be able to answer her question and how she can reach her colleague. Increased awareness leads to better understanding of the ongoing project. More well-directed communication improves the ongoing project. By increasing awareness and facilitating well-directed communication the goals of the FLOW Mapping technique are reached.

7 Case Study XP-Lab: Investigate Impact of FLOW Map

We selected one of the three FLOW techniques and evaluated it in a second case study. FLOW Mapping was selected since this technique is closely related to representing knowledge about communication. The main purpose of the FLOW Map is to increase awareness about communication in a distributed project.

7.1 Study Setup of XP-Lab

For the last six years, we offered an agile development class to Master-level students of computer science, called XP-Lab. They have learned about agile development practices when the class starts. In particular, they are familiar with eXtreme Programming (Beck 2000). In the agile development project, around 12-18 students are assigned to one or two projects. In the first phase of the semester, students learn to apply the practicalities of test-driven design, planning game etc. Tools and rules are presented and used. In the core part of the course students have to work full-time (approx. 40 hours) for one week. We call this week of intense agile development that is similar to industrial development *XP week*. During this time all agile practices are applied to our best knowledge and ability. For students the XP week is the most intense working and learning experience throughout the semester.

During summer term of 2010 we had a very special instance of this agile development class:

- There was a real customer with a real product vision. The customer wanted a
Smartphone application that combined a serious game on alcohol use and abuse, together with learning opportunities for teenagers and young adults. The outcome was supposed to serve as a demonstration prototype for a follow-up development.

- The project was geographically distributed between two collaborating universities: 4 students from Leibniz Universität Hannover and 4 students from Technische Universität Clausthal formed a distributed team. We had never used distributed teams before in XP-Lab.
- There was a project coordinator in Hannover who also acted as an XP coach for the distributed team. In an industry project, the tasks of this coordinator would be shared by project lead and XP coach.
- Agile practices were enforced during the entire semester. Both subteams in Hannover and Clausthal were supposed to follow agile practices and carry out communication at the high level of direct communication required by agile development (Cockburn 2002).

The customer travelled to the development site in Hannover and stayed there for most of the time. As an on-site customer, the customer was available for questions, discussions, and planning games. There is a two-hour driving distance between the sites at Hannover and Clausthal. Therefore, it was not feasible for the customer to visit both sites. Due to the asymmetric distribution of roles, Hannover became the head location, whereas the developers in Clausthal acted as a distributed extension of the workforce. Agile practices and the coordinator kept the teams together rather tightly.

On both sites the distributed team was equipped with:
- High-quality video-conferencing equipment for their exclusive use
- PCs, Subversion configuration tools, and dedicated Skype accounts
- Desktop sharing software to support development tasks
- Large SmartBoards that were temporarily used to display shared documents on both sides
- Shared mind map to manage story cards. User stories were denoted on story cards.
- Additional computer projectors for displaying data during a video-conference

Before the start of the project we had planned communication media and channels of information flow in accordance with the above-mentioned technical equipment. For example, during planning game sessions video conferences were supported by a distributed mind map for user story editing.

7.2 Execution of Case Study

It was one goal of this second case study (XP-Lab) to learn from the experiences of the pre-study. Therefore, we followed recommendations derived from the pre-study. For example, technical equipment (including Skype microphones, cameras, and other devices) was always tried out briefly before a communication needed it. This precaution may sound trivial, but has been violated in the pre-study as well as in many industrial projects. The preparation was surprisingly tedious and took a lot of effort (Meyer et al. 2010). On the other hand, poorly running technical equipment causes delays and annoyance and endangers success. This is unnecessary and should be avoided by our precaution.
The Clausthal students were invited to join the Hannover team for two sessions before the intense working week started. Getting to know each other in person is considered essential (Eckstein 2010). Since the intense phase of the project was only one week, more meetings in person were not needed or affordable.

We started out by planning the project based on a FLOW Map. The team and the communication channels to be used were planned. An initial map was created (see Fig. 12 center). A dedicated observer monitored changes. In the daily stand-up meetings, tasks and story cards were planned. Afterwards, the FLOW map was updated manually by changing the PowerPoint slide, exporting the image and uploading it to the project wiki. This was a tedious task. During the fourth development day we could not keep up with updating the map: Too many changes had occurred in a short period of time. At that point we had to stop updating the map for the rest of the XP week.

Fig. 12 depicts a snapshot of a FLOW Map that was created in XP-Lab. It shows the teams assigned to the two locations LUH and TUC. The actual map in the center of the figure was created before project start; it shows the desired information flows and their strengths (visualized by different line widths). Yellow pages information is given in the surrounding boxes (top and bottom). It includes participants’ names, pictures, roles, skills, and Skype IDs. Since the project was an agile project and Pair Programming was used, information about the members of pairs and their current tasks were added (bottom). Links to important documents were also added (top right) to ease access to them. A map with all associated information was constantly displayed on a large Smartboard in the developer room.

The developers had to fill in a questionnaire after each development day that contained the following questions according the usefulness of the FLOW Map:

- Did you use the FLOW Map or did you intentionally look at it today? (Answers: Yes | No)
Was the FLOW Map useful for you today? (Yes, a lot | Yes, a little. | No)
Was the FLOW Map up to date at all times today? (Yes | No).

Analysis of FLOW Map and questionnaire data led to some interesting findings.

7.3 Interpretation and Benefits of the FLOW Mapping Technique

A student project is not an industrial endeavor. In particular, student teams can better be controlled and observed, which is important for assessing the usefulness of FLOW Maps. Some findings were encouraging:

- We were able to plan desired information flows, and contrast it with actual information flow.
- A FLOW Map could be kept up to date manually for a while.
- The current status of communication and flows could effectively be fed back to the developers. Fig. 12 shows the planned or desired information flows. Fig. 13 provides the actual flows as observed. The display in the developer room showed the planned situation to encourage the developers to communicate with their distributed co-workers as desired.

However, manual updates are challenging during a period of frequent changes. As a consequence, we developed more tool support for (semi-)automated updates of the map (see http://flow-map.org).

When comparing planned (Fig. 12) and actual flows (Fig. 13) a number of deviations could be identified:

- More coaching than planned had been needed, represented by strong dashed arrows.
- Fluid communication between pairs worked better than expected. The labels next to the arrows indicate the main topic those communications addressed.
- There was intense flow of information from the customer - to both Hannover and Clausthal sites. This was not necessarily expected due to the set-up situation.

The role of solid documentation had been estimated to be low initially (Fig. 12). The actual FLOW map shows an even weaker role. It is characteristic of an agile project to rely more on direct communication than on documentation. The monitored FLOW map of actual information flows confirms this assumption. It shows that students managed to communicate directly despite the distribution and asymmetric set-up of roles in the project.

The FLOW map helped us to detect unexpected situations even during the one week of intense work. In the following two subsections we discuss the benefits of the FLOW Mapping technique in general and
usage of the FLOW Map in particular for the two main roles of the technique. According to Fig. 10 and Fig. 11 the main roles to be distinguished during FLOW Mapping are coordinator and developers.

7.3.1 Benefits for Coordinator

It was encouraging to see the applicability of FLOW Maps. As outlined above, they facilitated a comparison and provided some findings. Our questionnaires helped us to further investigate the usefulness and benefits of using a FLOW Map.

The main project coordinator was located in Hannover (the coach in Fig. 12 and Fig. 13). He carried out most of the coordination between Hannover and Clausthal. He especially coordinated communication between the two sites. He also provided support in agile approaches. He was particularly enthusiastic about seemingly subtle aspects of the FLOW Map. For example, visualizing the participating developers on both sites enabled him to call everyone by name from day one. This feature was available to all participants. Prominent yellow pages information allowed the coordinator to consider them and, thus, reach a level of higher awareness within short time.

The visualization of flows was also found useful. The coordinator preferred to look at the planned information flows rather than the actual ones during the project. The planned flows helped the coordinator to schedule and prioritize communication activities. Visualizing actual flow on the other hand is crucial for judging the status of the project quickly. It was beneficial for the coordinator to see the actual tasks assigned to all developers.

The role of a coordinator may not be filled in any given distributed project. In an industry project, the tasks of a coordinator could be carried out by the project lead or XP coaches. Both need in-depth knowledge and awareness of planned and actual information flows.

7.3.2 Benefits for Developers

Fig. 14 shows that most developers used the FLOW Map in the beginning of the XP week. During the first days the developers did not know each other's names and faces well and they had to get used to each other's working style (as motivated in the Section 1). Awareness of distributed project participants and knowledge of their communication habits were low in the beginning. This led to a usage rate of more than 75% in the first two days. On the third development day of the XP week the rate declined. On day 5 after the weekend the rate rose again.

We assume that the small size and short duration of the XP week was responsible for the decline in usage in the middle of the XP week. By the time they knew the team and had become familiar with their communication style there was no need to consult the FLOW Map anymore. Only after the weekend, when some awareness might have been lost, usage of the map rose a little bit again. In a larger project with more changes in structure and staff, the need for awareness information may last longer. Getting to know each other and other sites is a knowledge acquisition task typically carried out during the initial phase of a project.
In summary, the FLOW Map seems to have worked well during the initial phase of this short project. Awareness was increased in terms of knowing other team members, being aware of their names, tasks, and ongoing conversations. As we have outlined in the Introduction, acquiring knowledge and awareness is important in order to avoid misunderstandings and unnecessary friction.

8 Conclusions

Global software development requires companies to share knowledge on their values, processes, and on the way of letting information flow. There are numerous types of information flowing through a global software project: from requirements and design decisions to constraints, domain knowledge, and knowledge about the project. Participants need to learn about their partners. Like many other knowledge sharing tasks awareness-building activities need to be lightweight for project participants. Techniques and tools are needed to facilitate them.

We present FLOW and the FLOW Method for managing knowledge about information flow. Direct communication and documentation contribute to information flow. The first step towards optimizing information flow is to capture, visualize, and model it. When we look at a project from the perspective of information flow certain aspects are easier to see: Direction and mode of information transfer and the distinction between solid (documented) and fluid (non-documented) flows of information. We illustrate the FLOW perspective in two case studies and an intermediate construction of FLOW techniques.

- In a pre-study a globally distributed project was analyzed for information flow problems. Focusing on flows as a potential reason for problems helped us to identify three recurring problems. The identified problems were missing context knowledge, missing solid information,
and missing awareness. Observations collected from the pre-study shed some light on the kinds of problems missing context, missing documents, and missing awareness can create. These three examples show that it is helpful to use the FLOW perspective in a distributed setting.

- In a next step we designed FLOW techniques for each of the three problems using the FLOW Method as a framework. Of course, the FLOW Method is not restricted to the three examples presented here.

- In a second case study we applied and evaluated usage and benefits of the FLOW Mapping technique and FLOW Maps. Based on the FLOW Method, FLOW Mapping was specifically designed to improve awareness and communication in GSD projects by visualizing information flows in a FLOW Map.

Although we used students in our case studies we think our results are also relevant for industrial projects, because we believe that communication problems (e.g. awareness problems) are not much different, maybe even worse, in real world projects and thus our approach could be even more useful there.

Encouraging findings showed that FLOW Maps can be used to plan and maintain communication, and can create benefits for project managers (e.g. project leads or coaches) as well as for developers. While an in-depth discussion and evaluation of further FLOW Techniques is beyond the scope of this paper, we consider FLOW Maps a case for demonstrating feasibility and potential further benefit of applying our FLOW Method.

Further research is needed to investigate mid- and long-term effects. Changes occur, and developers may start forgetting about other sites. At this point, the awareness-increasing benefit could last longer. One might see new benefits of FLOW Maps that could not materialize in our case study. Manual effort overwhelmed our observer, despite the short and small project. Therefore, we started to develop a tool for monitoring many aspects of information flow automatically or at least with minimum effort manually. Kwan et al. (Kwan et al. 2006) proposed a similar tool for requirements-centered social networks.

However, information flows cannot be completely inferred from automatically monitored communication events. From a theoretical point of view information flows and communication acts are not equal. Automatically monitored communication acts can be used to check whether there was a possibility of desired information flowing. On the other side, personal family chat can lead to long conversations (over phone, Skype, or e-mail), but will not amount to project-related information flow. The following points need to be considered whenever tools are created for automatic capturing of information flows and interpretation of automatically collected communication events:

- Automatic monitors can only monitor communication events. It is very difficult and most of the time unfeasible to infer the content of these communications.

- Information flow models developed for planning or analysis do not contain singular communication acts. Fluid communications are only drawn when they are important, recurring, and project-related. It is not always easy to draw the line between random chat and professional conversations.

Improvement requires change. From the FLOW point of view, special techniques can be created to improve selected aspects of information flow. The FLOW Mapping technique is a useful tool for planning and observing information flows in distributed
projects. We could show that it is especially helpful for supporting the coordinator in setting up the project, as it supported him in building a team. Developers in a distributed software project appreciated the FLOW Map especially in the beginning.

In an industrial environment, simple and easy-to-use support techniques are essential for effective improvement. Our FLOW Method is based on a notation that was designed and optimized for simplicity and focuses on the key concepts of fluid and solid information, i.e. the co-existence of knowledge in documents and in people. This contribution aims to present an innovative new perspective on these crucial issues. The Method is designed to reach from analysis of a given project situation down to concrete and tailored improvement techniques.

In summary, we presented FLOW and the FLOW Method as a means of managing knowledge about communication in software development projects, which is especially important in global settings.

Acknowledgement

We would like to thank the students and researchers who helped to plan, execute, and evaluate the case studies. This work was funded by the German Research Foundation (DFG project InfoFLOW, 2008-2011).

References

Stapel, K., Knauss, E. & Schneider, K. (2009), Using FLOW to Improve Communication of Requirements in Globally Distributed Software Projects, in 'Workshop on Collaboration and Intercultural Issues on Requirements: Communication, Understanding and Softskills (CIRCUS 09)', Atlanta, USA.
Wolf, T., Schröter, A., Damian, D. & Nguyen, T. (2009), Predicting Build Failures using Social Network Analysis on Developer Communication, in '31st International Conference on Software Engineering (ICSE’09)'.

© 2010 Blackwell Publishing Ltd