Using FLOW to Improve Communication of Requirements in Globally Distributed Software Projects

Kai Stapel, Eric Knauss, and Kurt Schneider
Software Engineering Group, Leibniz Universität Hannover
{kai.stapel, eric.knauss, kurt.schneider}@inf.uni-hannover.de

Abstract

Effective communication of requirements is essential in all software projects. In distributed projects communication faces even more and bigger challenges due to the narrow communication channels that are available. Applying information theory and information flow concepts as provided by FLOW to process improvement can help to overcome challenges associated with communication of requirements in global software projects. We participated in a software project distributed over three locations in three time zones. To test the feasibility of our FLOW perspective we analyzed the projects information flows. We found communication problems that are typical in a global context. Based on FLOW theory we developed three new suggestions to overcome these problems. Finally, we present a measurement approach that can be used to evaluate our suggestions in future global projects.

1. Introduction

Global Software Engineering research showed that many distributed projects face communication problems [1-4]. In a global context collaboration is restricted because of limited communication channels [3-5]. This leads to communication breakdowns, misunderstood requirements, and finally to project delay or failure. Solutions have been proposed and evaluated to overcome communication problems in distributed development [4-6]. Although quite a few contributions found informal communication to be a main driver of (global) software projects [1, 2, 5, 7-9], not many approaches have been proposed to explicitly take informal, verbal, or ad-hoc communication into account. But, especially at project start, when distributed teams still grow together and requirements are unclear, these types of communication are very common [1, 5, 7, 10] and thus should be analyzed and optimized.

Therefore, we present our FLOW approach that allows considering these common but hard to capture types of communication during (1) project analysis and (2) project optimization. In project FLOW we develop techniques and tools to overcome the difficulties associated with capturing, analyzing, and improving informal along with other flows of information.

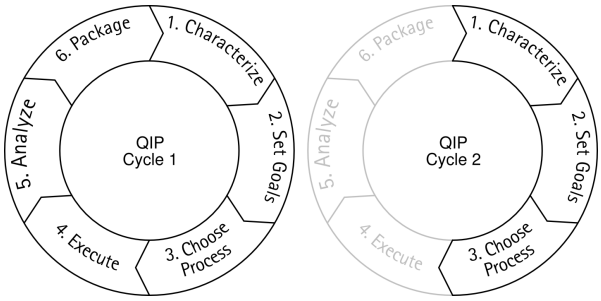

Figure 1. Status quo in two consecutive QIP cycles as of this writing

The systematic improvement method to guide our efforts in evaluating the applicability of FLOW to improve communication of requirements in distributed software projects is the Quality Improvement Paradigm (QIP) [11]. According to Figure 1, six steps need to be carried out in order to achieve goal-directed improvement. Each QIP cycle starts with characterizing the problem, setting goals for the improvement, and observing the process while it is being executed. It is well-suited for our purpose since it also emphasizes that observations need to be analyzed, packaged, and fed into the next improvement cycle.

We started with analyzing a distributed software project between Finland, Russia, and Germany. Based on this pilot study we were able to show that well-known problems of distributed software development can be captured with the help of the special perspective provided by FLOW. After that, we used FLOW theory to identify new approaches to overcome these problems. As of this writing we are done with planning the measurement for evaluation of the suggested innovative approaches in future distributed projects.

Stapel, K., E. Knauss, and K. Schneider. *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)*. 2009. Atlanta, USA.

This paper is structured as follows. Section 2 gives a short overview of related work on communication issues in global software engineering and proposed solutions to overcome them. The basis for all other sections are the FLOW fundamentals in section 3. Sections 4 to 6 are descriptions of the QIP phases as depicted in Figure 1. In section 4 we give a short description of the observed project, our measurement goal for the first QIP cycle, the tools we used for measurement, project execution, and analysis results. In section 5 we discuss conventional solutions to the observed problems and compare them to more innovative approaches, motivated by FLOW theory. In section 6 we show how to evaluate the presented approaches in future distributed projects to prove or disprove their effectiveness.

2. Related work

The impact of good developer communication on software quality has been discussed frequently. Wolf et al. were able to give objective evidence of the importance of good communication [9]. Based on analysis of social networks they were able to predict failures in the integration builds of a large highly distributed software project.

In [4], Bird et al. show that it is possible to achieve comparable software quality in distributed development as in collocated software development. The prerequisite for this is to overcome well-known challenges of distributed development: Loss of communication richness, coordination breakdowns, geographic dispersion, and cultural differences [3]. In [4], 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 artifact ownership, and (e) organizational integration.

Battin et al. [6] suggest three similar solutions to overcome the same challenges: (a) liaisons, (b) distribute entire things for entire lifecycle, and (c) plan to accommodate time and distance.

Herbsleb and Grinter [5] offer six solutions to overcome problems associated with distance. Three 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 are proposed to improve 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.

Bruegge et al. [8] suggest a tool that encourages developers making communication about system models, collaboration artifacts, and organizational models explicit. They observed that usage of this tool increased awareness of relevant stakeholders and thus enables informal collaboration.

These studies - like many others - emphasize the importance of communication issues. 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 suppose that this is due to the difficulties associated with capturing and modeling this kind of information flows consistently with other communication relationships. Our concepts to overcome the difficulties in capturing, analyzing, and improving informal communication are being developed in project FLOW [12].

3. FLOW

In project FLOW we systematically investigate information flows in software development. We consider all types of information flows during software process analysis and improvement, particularly verbal and informal communication. This is a differentiating characteristic of FLOW with respect to many other process or workflow improvement approaches.

To highlight the main characteristics of information flows a metaphor of state of information is used:

- *Solid information* refers to information that is (1) long term accessible, (2) repeatable, and (3) comprehensible by third parties.
- *Fluid information* is defined as all other information that violates one of the above criteria.

Typical representatives for solid information are books, formal documents, formal e-mails, or source code. Fluid information is knowledge in peoples' minds or informal notes, e-mails etc. that require preknowledge (context) for correct interpretation. Requirements are fluid until they are solidified in a specification.

Along with solid information any software development heavily relies on fluid information flows [13]. Fluid information is difficult to capture and difficult to analyze. Therefore, we assume that distributed projects can better be analyzed and improved when special techniques for analysis of fluid information flows are incorporated in the software improvement process.

The following two sub-sections describe aspects of FLOW that are theoretical foundations for the remain-

der of the paper. For a more thorough description of FLOW see [12, 14].

3.1. FLOW notation

A notation is needed to model and depict information flows. FLOW models are mainly used for discussion, so the FLOW notation was designed to be simple and easy to use. Therefore, it consists of only a few symbols.

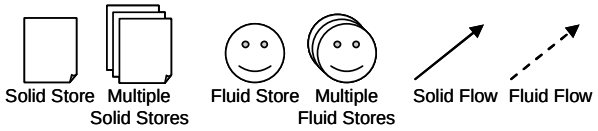

Figure 2. Basic FLOW notation

A document symbol depicts a solid information store; multiple documents depict multiple solid stores. A smiley symbol depicts a fluid information store; multiple smileys depict multiple fluid stores, e. g. a group of people. Solid information flows, i.e. flows originating from a solid store, are depicted by a solid arrow. Fluid information flows, i. e. flows originating from a fluid store, are depicted by a dashed arrow.

3.2. Information, knowledge, and context

FLOW theory provides an explanation for why communication among globally distributed partners is so difficult. This explanation is rooted in the nature of information, knowledge, and their transfer between people – also known as communication.

Knowledge is the capability and skill of a person to solve a problem. Knowledge is, therefore, bound to individuals [15]. In order to share knowledge it needs to be externalized as information.

Information is data plus meaning [16]. When sharing knowledge between two people the knowledge source has to "create" the data and transmit it to the knowledge target. The data can be written or spoken words, for example. The target person needs to receive and understand the data. The process of understanding is the tricky part. To associate the correct meaning to the data the original context needs to be known. In this sense *context* refers to all prior knowledge a person has to have in order to correctly interpret given information. It ranges from cultural background and language to specific domain knowledge or low level project details.

Context is usually not shared extensively during a professional conversation. Context is rather assumed to already be in place. This may hold for collocated teams most of the time, where people have worked together for a long time and came to know each other well. In globally distributed teams members come from different cultural, educational, and professional backgrounds. Hence, only a very small common context can be assumed. Dealing with this implicit separation places a significant burden on communication in distributed projects.

We define *project context* to be all prior knowledge that is needed to correctly interpret information relevant for a project. Project context consists of a lot more knowledge than plain knowledge about the project. Usually it contains knowledge about legacy systems, customer preferences, market conditions, domains, processes, habits, conventions, and corporate culture.

The following example illustrates the relations between the terms knowledge, information, and context. Imagine two developers who successfully developed a tool for agile project management support, revision A collocated at their American office. In a new project they have to develop revision B of the tool. The goal is to fix the main drawback of revision A, the missing version control integration. Developer A writes developer B an e-mail that they first need to change the file format from a binary to a text based type. Developer B knows that CVS is the only option for version control since their main customer cannot use any other system for legacy reasons. Thus, he correctly interprets the information in the e-mail and infers the requirement for a text based file format in revision B. Developer B had sufficient context to understand the e-mail.

To speed up development of revision B a developer located in Germany is integrated in the team. She does not know product revision A nor customers from the American office. American developer A forwards the e-mail with the file type change requirement to the German developer. Since she does not have prior knowledge about the customers constraints she misinterprets the mail and starts to change the file type to a word processor type, which still is binary. Nobody from the American department notices that until four phone conferences later. The two weeks of development the German developer has spent were all useless. The German developer's project context was not sufficient to understand the requirement.

Thus, any developer in a software project needs to have as much project context as possible to reduce broken information flows and misinterpretation.

4. Pilot study: Analyzing a global project

In the first QIP cycle (see Figure 1) we analyzed a globally distributed project named Minotaurus from the

FLOW perspective. The goal was to better understand global software development. In this pilot study we did not plan to incorporate any suggestions for improvement in the running project. Therefore, the following (but one) sections about the phases of the QIP cycle only contain descriptions of our observation method, not the project under observation. A summary of the project itself is given upfront in the description of the first QIP phase.

QIP-1.1. Characterize: Project Minotaurus was a globally distributed project joining developers from the three countries Finland, Russia, and Germany. Each site participated with 3 developers. The project lead was in Finland. Developers from Russia and Finland had worked together in previous projects. The German developers were new to the global team. All developers were graduated students with at least 3 years experience in software development.

The goal of the project was the extension of the open source project management tool Trac with special features for agile development. The programming language was Python. Source of requirements were the research group at VTT, Finland and two previous versions of the software.

A SCRUM based process was used for project management and coordination. Scrum Master and Product Owner were located in Finland. The backlog was maintained in a Wiki. Trac 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 week represents one iteration, i. e. one sprint. It is important to know that the German team entered the project one week after start and they left after two iterations. The results presented in this paper are mostly based on the data gathered during the two weeks of the German participation. A video summarizing communication aspects of the project is available at www.se.uni-hannover.de/en/glose/minotaurus.

QIP-1.2. Set goal: Understand global software development from a FLOW perspective. The goal of the first QIP cycle was to understand global software development and, if necessary, identify areas for improvement based on FLOW theory. To reach that goal we set up measures that were specifically designed for the information flow perspective in global projects.

QIP-1.3 Choose process: Observation packages. Our approach for identifying the specifics – especially specific problems – in global software engineering was to capture the experiences of the developers during the project with a special focus on information flows. Our (manual) tool for capturing experiences is a so called observation package [17]. An observation package is a simple form with fields for each part of an experience: (a) the observation, (b) an emotion, and (c) a conclusion or suggestion. Pre-defined multiple choice check boxes make it easy to classify an observation on the fly to be about one or more of the following: human factor, communication, information flow in general, media, or technology related. Another classification can be made for the type of emotion the observation relates to. It can either be a problem, something interesting, typical distributed, or a good idea.

We asked every developer to fill out an observation package whenever she experiences something that is related to distributed development. An example of a filled out observation package is given in Figure 3.

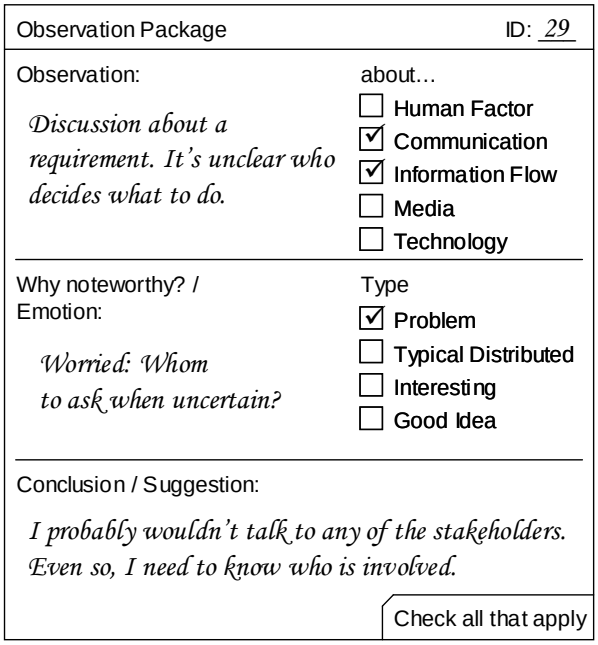

Figure 3. Example observation package.

Besides collecting observation packages we also recorded audio, video, and screens during conference calls. A third data source for analysis was the monitoring of communication by a dedicated observer. He logged time, duration, involved participants, roles of participants, used media, state of information flow, and rough content of communication events.

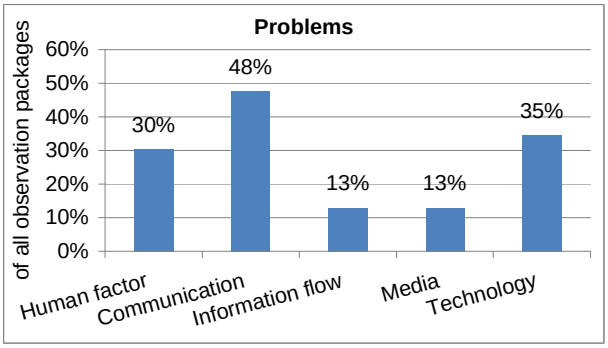

Figure 4. Distribution of observed problems

QIP-1.4. Execution: During the two weeks of the German participation in project Minotaurus the developers filled out 44 observation packages. 846 minutes of communication were monitored. Unfortunately, the observer could not monitor the entire project, due to other duties. Together with the Russian and Finnish data a total of 280 Gigabytes of audio, video and screen recordings were collected.

QIP-1.5. Analyze: Focusing on communication. Analysis of the observation packages showed that most of the observed problems were related to communication. Figure 4 shows the distribution of the observed problems in relation to all 44 collected observation packages.

This finding was not surprising in the context of distributed development [1-4]. We were encouraged by the fact that our own observation confirmed some of those findings. Selected examples of observed problems are given in Table 1, Table 2, and Table 3.

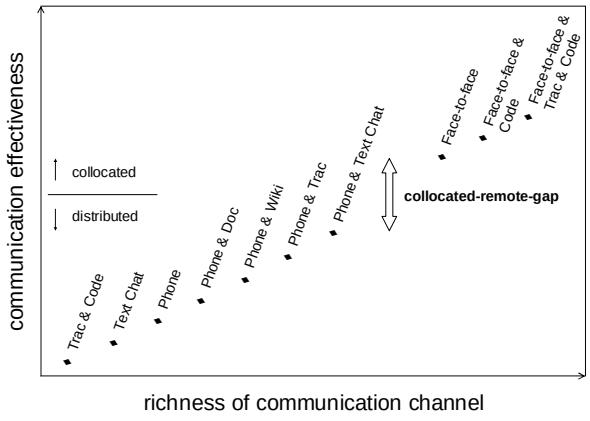

Figure 5. Communication channel width. Chart in the style of [13].

Figure 5 depicts the types and combinations of different communication channels as logged by the communication event observer. The types are related to each other in terms of richness of communication channel and communication effectiveness in the style presented by Cockburn [13]. The diagram shows the gap between local and distributed communication effectiveness. This gap is the cause for distributed communication being so problematic, especially when it comes to context sharing. Both axes are ordinal scales. We claim that one media is more effective or richer in terms of channel bandwidth than another, but neither Cockburn's [13] nor our findings are sufficient to constitute quantitative relationships. For example, although face-to-face communication has been confirmed to be more effective and rich than phone calls, we would not argue whether it is twice or 1.5 times as effective.

5. Packaging observations (QIP-1.6)

The final part of a QIP cycle is to consolidate the experience gained and package it for reuse in future projects. Our packaging approach was to first find solutions for communication problems that already have been proposed [4-6, 8, 18] (see related work) and contrast them to novel approaches that we developed based on FLOW theory [12, 14]. We packaged suggestions for improvement to the three problems of (5.1) missing context for interpretation of requirements, (5.2) missing awareness, and (5.3) missing documented information.

5.1 Ambassador as a context sharing catalyst

In distributed projects, especially if teams are participating that did not work together before, only a very small common project context is given (see 3.2). The missing context leads to misinterpretation of information or information flow breakdown, i. e. a flow does not reach its target. This happens if the target ignores the received information, because she does not know that it is relevant, or if it simply does not understand it. Some of our observations reflect that.

Table 1. Obs. related to missing context.

ID	Observation and emotion part of experience
6	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.
14	Finnish and Russian developers talk about last iteration. The Germans cannot follow. They are afraid they missed important information.

A common project context cannot develop as fast as in collocated teams because of the restricted communication channels (see Figure 5). Therefore, the goal of

this suggestion is to speed up development of a larger common project context particularly at the beginning of the project, i. e. the requirements phase.

Obvious solution: Aim for collocation. A trivial and rather expensive strategy to avoid the collocated-remote-gap and therefore overcome communication problems caused by missing context is to aim for collocation as much as possible. Initial meetings of the teams should be held collocated. The customer should meet in person with the requirements analysts. A so called ambassador [18] or liaison [6] should be sent to a remote project site to improve understanding between the global teams.

Suggestion from FLOW perspective: Send context sharing ambassador. The trivial solution will already improve communication in a distributed project, but it is expensive and not always feasible. So our approach is to concentrate on the missing common context. Thus, our suggestion is to use an ambassador as mentioned above, who especially acts as a "context sharing catalyst". To increase the common project context 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
- Rational behind decisions
- Well established sources of information like Intranets, Wikis, or other experience factories.

To achieve that an ambassador should socialize with all project members and stakeholders. Being a gregarious person as suggested by [5] will be beneficial for that. A context sharing ambassador should:

- Attend meetings regularly.
- Report back to home site regularly.
- Negotiate problems between distributed members of the team or at least attend negotiations.
- Be a context scribe (see suggestion in section 5.3.). The context scribe's task is to explicate foreign project context and document it, e. g. in a Wiki.

5.2 Increase awareness with FLOW maps

Developers cannot consider issues they are not aware of. Opportunities to exchange information and discuss requirements should be used to avoid misunderstandings. Exchanging information requires good orientation in a distributed project. It also requires good awareness of the distributed team. 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 emphasize this very strongly:

Table 2. Observations related to awareness.

ID	Observation and emotion part of experience
29	Developers discuss about a requirement. It is unclear who decides what to do. They were worried whom to ask when uncertain.
39	After a Skype conversation two developers reported that they were not sure who had been their conversation counterparts. They felt very confused about that.
44	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.

Also, our video recordings showed that many times one team was not even aware who was currently present in the development room at the two other locations and what tasks those people were working on. Real group awareness can only emerge when developers know who constitutes the group, and who may be reached at a given point in time.

Obvious solution: Video links. Audio and video links seem to be the most straight-forward solution to the problem of basic unawareness, as described above. Broadcasting video streams along with audio among the project members improves awareness of who is present. However, permanent video connection between sites can be a problem when network connections are slow or limited. In such a situation, a webcam with less frequent update rates can be an approximation.

Suggestion from FLOW perspective: Use FLOW maps. The lack of awareness on the basic level usually leads to high barriers for direct contacts and to impoverished communication between distant sites. While there is rich informal communication within each site, exchange between distributed individuals and teams is limited.

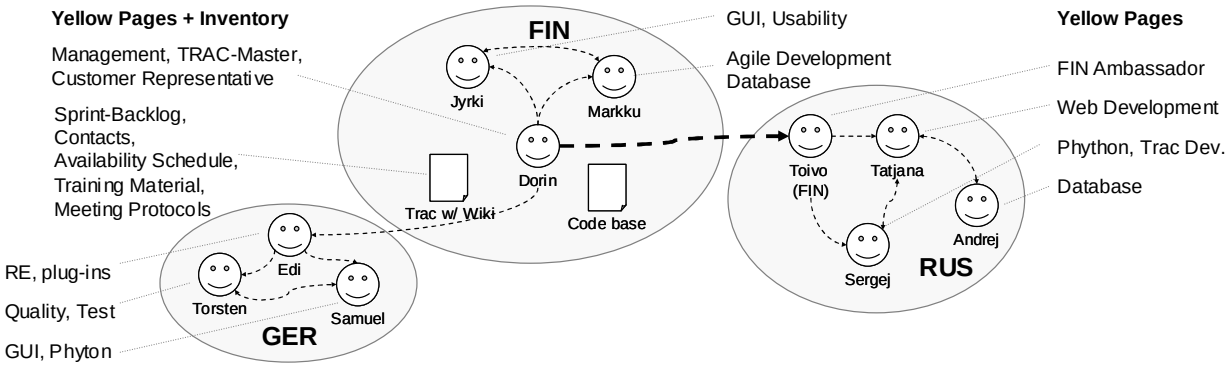

Figure 6. Yellow Pages combined with FLOW map of main fluid flows of requirements.

In order to make everyone aware of all others on the project, a list of all participants can be created, e. g. during or after the kick-off ("Yellow Pages"). It should contain names, phone numbers, e-mail and Skype addresses, and a few hints about the tasks and competencies of a person. Adding a photo can further increase awareness. However, the Yellow Pages alone are not sufficient to trigger point-to-point communication. In project Minotaurus we already had one in place and still made the above observations.

To stimulate point-to-point communication we suggest to model flow of requirements and other relevant information in a FLOW map (see Figure 6 and Figure 7). The goal is to explicitly focus on *awareness about existing and desirable information flow*.

1. Draw a fluid store (face) symbol for each known participant. Large projects might need to use several hierarchical levels.
2. Draw solid store (document) symbols for all documents that are essential to exchange the kind of requirements or information you are focusing on.
3. Sketch a dashed line between two people if they exchange requirements on a regular basis. Put an arrow if the information tends to flow in one direction rather than the other.
4. You may use different line width to emphasize frequent and intense flows of requirements. In Figure 6, Dorin tends to talk to Toivo a lot.
5. It will be helpful to put an extract of the Yellow Pages and Inventory on the side of the FLOW map. Omit details and contact info that would clutter the overview.
6. Print the map on a large piece of paper and put it up in the development room.

Note that not each and every communication act needs to be modeled. Since only major and recurring flows of requirements are relevant for awareness of requirements flow, single or accidental communication is not shown. Most conversations seem to be two-way exchanges. However, the FLOW map is supposed to abstract from the exchange of words and illustrate only the rough direction of requirements. In Figure 6, for example, Dorin might have video or Skype conferences with Toivo and Edi. All three will talk, but Dorin – as the customer representative – is the source of requirements. Within each time, flows are different.

Since we do not attempt to give a detailed and accurate account of all communication acts, FLOW maps do not need automatic screening capabilities (such as proposed by Damian et al. for RCSNs [19]). An overview FLOW map represents the perception of the developers drawing it. Each site may perceive the situation slightly different.

A FLOW map emphasizes fluid flow of information – an aspect that is neglected in most other representations of a distributed project. By visualizing people and major fluid flows, the FLOW map provides orientation and might cause participants to reconsider communication habits.

Figure 6 represents the situation as we saw it in Minotaurus. It is an important step towards awareness and effective requirements propagation to model information stores and containers as they are. However, improving flows based on improving awareness is one more important step.

Constructive use of FLOW maps starts by reusing the model of the existing situation. A certain focus is defined. For example, Markku (FIN) recognizes he should talk to Andrey (RUS) about the interpretation of a database requirement. According to Figure 6, Markku would have talked to Dorin, who might have mentioned the issue to Toivo in there next Video meeting. Toivo (FIN-ambassador to RUS) communicates with Andrej only indirectly due to language barriers. Markku's information flows through Dorin-Toivo-Tatjana, before it reaches Andrej. This is the Chinese whisper pattern. Fluid information flowing over several indirections tends to get distorted or lost. According to Figure 6,

there is no established flow back. When Markku gets aware of that situation, he can propose a direct phone call to Andrej. However, since Andrej feels uncomfortable in abstract English conversations, they agree on using Skype while they both look at the code relevant for their requirement. These improved information flows are modeled in Figure 7. Irrelevant flows are faded out to reduce visual complexity.

Different concerns and situations may call for different FLOW maps, which all should be kept consistent with the orientation overview when new flows will have established (updating fluid flows in Figure 6). FLOW maps are cognitive tools to increase awareness by focusing on requirements flow. They should be updated for example at each iteration start.

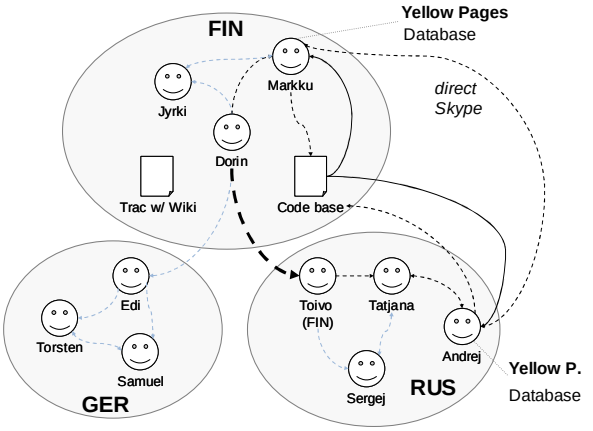

Figure 7. Improving flows on database issues.

5.3. Solidify information as a by-product

Several of the observed problems clustered around fluid information that was forgotten or lost. Missing information caused significant confusion and frustration among developers.

Table 3. Observations about missing infos.

ID	Observation and emotion part of experience
18	Russion developers were searching for some requirement that had been mentioned in a distributed meeting. They could not find it on Subversion.
10	Trac tickets were out of date. Documented and discussed status diverged.

Obvious solution: Document more. There is a trivial recommendation for avoiding most problems that result in fluid information being lost or forgotten: mode information should be solidified, i.e., documented. Fluid information tends to "run through the fingers" of those who hold it.

There is justified objection to that recommendation. Documentation takes a lot of time and effort both for writing and reading, which may slow down development. Hence, developers are often averse to documentation.

In a slightly more efficient set-up, a dedicated scribe or librarian may be in charge of taking notes and documenting essential information: contact information, availability schedules, decisions on requirements, responsibilities assigned in a meeting, project status, etc. Managerial and control information is important to solidify and store in a common repository, but requirements and technical details also need to be memorized, and hence solidified.

Suggestion from FLOW perspective: Solidify as a by-product. Our key concept in overcoming the solidification bottleneck is the concept of solidifying information as a "by-product" of an activity that takes place anyway. In [20], the *"By-product approach"* for capturing rationale in RE is described. Its basic idea is to build a dedicated tool with specialized recording facility. This tool is used during a mainly fluid meeting or conversation, and transforms fluid flows into solid records. A record may not be a Word document, but it is *solid* by the criteria defined above.

The feasibility and efficiency of that approach depends on "deep recording": A simple audio tape is a solid record of a phone call or a software demo. However, it is tedious to find pieces of information in a non-indexed audio tape of some length. Therefore, an appropriate by-product tool needs to provide automatic indexing together with recording. Examples of by-product tools for capturing decisions in a software prototype demonstration are given in [20, 21].

Starting point for building by-product tools in a distributed setting is to use the digital media that is used for communication anyways as a basis for recording and indexing.

6. Towards improved communication of requirements in global SE (2nd QIP cycle)

In the first QIP cycle we were able to detect the following communication problems with the help of the perspective provided by FLOW: small common project context because communication channels are not rich enough, members of the distributed team are not aware of what is going on in the project, and information gets lost across distribution borders. During packaging (QIP-1.6) new approaches to overcome these problems of distributed software engineering were developed. In

the second QIP cycle we plan to measure the effectiveness of these new approaches.

QIP-2.1. Characterize: The project that serves as the basis for measurement in the second QIP cycle should be similar to project Minotaurus to assure comparability. It should have at least three participating sites, a total of 10 medium experienced developers, and between one and three months duration. The development style should be agile.

QIP-2.2. Set goal: Measure effect of FLOW based proposals. The following abstraction sheets [22] illustrate our evaluation strategy for each proposal.

Goal 1: Measure effect of context sharing ambassador on common project context in distributed development.	
Quality Factors	**Factors of influence**
# of misinterpreted info # of info flow breakdowns	Activity of ambassador Character of ambassador
Hypotheses	**Influence on Hypothesis**
# of misinterpreted info decreases by 50% # of breakdowns decreases by 50%	Ambassador unifies / increases common context at project start. Ambassador's ability to socialize

Goal 2: Measure effect of FLOW map on awareness in distributed development	
Quality Factors	**Factors of influence**
Effort to find responsible developer in dist. team. # of bilateral communication early in the project	Level of detail of FLOW map Timeliness of FLOW map (up-to-date).
Hypotheses	**Influence on Hypothesis**
Developers find it easy to identify peers that work on similar topics # of bilateral comm. raises by 50 % in first 2 weeks.	FLOW map gives a fast overview of who knows what. FLOW map shows current state.

Goal 3: Measure effect of documentation as a by-product to overcome information drain	
Quality Factors	**Factors of influence**
# of lost information Effort to create solid infos Effort to access solid infos	Specialized FLOW tools • for domain • for GloSE context
Hypotheses	**Influence on Hypothesis**
# of lost infos decreases 50% Effort to create documentation increases 25% Effort to access infos decreases by 50%	Specialized by-product tools help to keep documentation effort low and help to find relevant information fast.

QIP-2.3 Choose process: To evaluate these solutions and observe informal communication in the required level of detail, we need to create some infrastructure and instrumentalization for the next project.

First, all suggested approaches should be set up. Ambassadors should be sent to remote sites. Preferably they should be gregarious people. FLOW maps should be created and put up in the development rooms. The maps should be updated at start of each iteration. A scribe should be appointed to solidify important information, e. g. requirements. If there is enough time for preparation an alternative or addition to the scribe would be the creation of a dedicated by-product tool for global software development, ideally one that helps documenting requirements as they are discussed in distributed meetings.

Observation challenges and future work. Detailing the abstraction sheets down to concrete measures, however, remains a challenge. How can context – in particular common project context – be measured? How can the number of misinterpreted or broken information flows be measured? How can awareness be measured? How can the right amount of documentation be measured? As a first approximation the number of observation packages that mention problems related to missing context, missing awareness, or missing documentation can be used. In addition, observers that are trained in FLOW analysis can be sent to each development location, or already present developers can be quickly trained before project start [10] to capture the relevant information flows remotely. A combination of a survey based approach [10] and Social Network Analysis [9] could be used to measure team awareness.

7. Conclusion

In this contribution we presented an approach for analysis and improvement of global software projects based on the principles of our FLOW concept. We structured our efforts according the QIP. In the first QIP cycle we analyzed a distributed agile project with developers from Finland, Russia and Germany. Analysis of the 44 collected observation packages showed that most problems of global development were communication related. The three main problems we identified were: (1) missing context for interpretation of requirements, (2) missing awareness, and (3) missing documented information.

Based on these findings and FLOW theory we developed options for improvement. We claim that a large common project context is important for distributed development – independent of the development methodology, whether it is agile or plan driven – and that it can only be increased with fluid information ex-

change. Therefore, we propose to send ambassadors to remote project sites who improve fluid information transfer between sites through their presence and their special tasks dedicated to context sharing. Although there certainly are a lot of information sources and flows in global software engineering, many developers are not aware of the important ones. This hinders effective point-to-point communication and leads to Chinese whisper like chains of information flow. A FLOW map helps to increase awareness about who knows what, where the important information flows are or should be, and where requirements come from. Thus, such a map can trigger direct information flows. A lot of information is flowing fluid. Some of it should be solidified so it can be accessed again and reused for others. We propose the by-product strategy to enable solidification of information while keeping documentation efforts low.

The final step is to prove these novel suggestions effective. Therefore, we started to plan the evaluation in a future global software project. An approach for measurement has been proposed based on goal definitions and abstraction sheets. With that the second QIP cycle has been started. We call for discussion, application, and evaluation of our proposals in other's distributed projects.

If our suggestions can be proved to be effective, we could show that FLOW concepts can be used to improve communication, especially communication of requirements in a global setup, since it heavily relies on fluid information flows.

Acknowledgement. Many thanks to the developers and researchers of project Minotaurus from VTT, Finland, the Petrozavodsk State University, Russia, and the Free University of Bolzano, Italy.

This work was funded by the German Research Foundation (DFG project InfoFLOW, 2008-2011).

References

[1] Curtis, B., H. Krasner, and N. Iscoe, *A Field Study of the Software design Process for Large Systems.* Communications of the ACM, 1988. 31(11): p. 1268-1287.
[2] Grinter, R.E., J.D. Herbsleb, and D.E. Perry. *The geography of coordination: dealing with distance in R&D work.* in *International ACM SIGGROUP Conference on Supporting Group Work*. 1999. Phoenix, Arizona, USA: ACM.
[3] Carmel, E., *Global Software Teams: Collaborating Across Borders and Time Zones*. 1999: Prentice Hall.
[4] Bird, C., et al. *Does Distributed Development Affect Software Quality? An Empirical Case Study of Windows Vista*. in *31st International Conference on Software Engineering*. 2009. Vancouver, Canada.
[5] Herbsleb, J.D. and R.E. Grinter, *Architectures, Coordination, and Distance: Conway's Law and Beyond*. IEEE Software, 1999. 16(5): p. 63-70.
[6] Battin, R.D., et al., *Leveraging Resources in Global Software Development.* IEEE Software, 2001. 18(2): p. 70 - 77.
[7] Herbsleb, J.D. and A. Mockus, *An Empirical Study of Speed and Communication in Globally Distributed Software Development.* IEEE Transactions on Software Engineering, 2003. 29(6): p. 481 - 494.
[8] Bruegge, B., A.H. Dutoit, and T. Wolf. *Sysiphus: Enabling informal collaboration in global software development*. in *IEEE International Conference on Global Software Engineering*. 2006. Costão do Santinho, Florianópolis, Brazil.
[9] Wolf, T., et al. *Predicting Build Failures using Social Network Analysis on Developer Communication*. in *31st International Conference on Software Engineering (ICSE'09)*. 2009. Vancouver, Canada.
[10] Stapel, K., E. Knauss, and C. Allmann. *Lightweight Process Documentation: Just Enough Structure in Automotive Pre-Development*. in *EuroSPI²*. 2008. Dublin, Ireland.
[11] Basili, V., G. Caldiera, and R.H. D., *Experience factory*, in *Encyclopedia of Software Engineering*, J.J. Marciniak, Editor. 1994, John Wiley & Sons: New York. p. 469-476.
[12] Stapel, K. and K. Schneider. *Project FLOW*. Software Engineering Group, Leibniz Universität Hannover. June 18, 2009, http://www.se.uni-hannover.de/en/Research/FLOW.
[13] Cockburn, A., *Agile Software Development*. 2002: Addison Wesley.
[14] Schneider, K., K. Stapel, and E. Knauss. *Beyond Documents: Visualizing Informal Communication*. in *Third International Workshop on Requirements Engineering Visualization (REV 08)*. 2008. Barcelona, Spain.
[15] Probst, G., S. Raub, and K. Romhardt, *Wissen managen: Wie Unternehmen ihre wertvollste Ressource optimal nutzen*. 5 ed. 2006: Gabler.
[16] Floridi, L., *Is Semantic Information Meaningful Data?* Philosophy and Phenomenological Research, 2005. LXX(2).
[17] Schneider, K., *Experience and Knowledge Management in Software Engineering*. 2009, Berlin: Springer.
[18] Braithwaite, K. and T. Joyce. *XP Expanded: Distributed Extreme Programming*. in *6th International Conference on Extreme Programming and Agile Processes in Software Engineering*. 2005. Sheffield, UK.
[19] Damian, D., S. Marczak, and I. Kwan. *Collaboration Patterns and the Impact of Distance on Awareness in Requirements-Centred Social Networks*. in *15th IEEE International Requirements Engineering Conference (RE 2007)*. 2007. New Delhi, India.
[20] Schneider, K., *Rationale as a By-Product*, in *Rationale Management in Software Engineering*, A.H.M. Dutoit, R.; Mistrik, I.; Paech, B., Editor. 2006, Springer: Berlin, Heidelberg. p. 91-109.
[21] Schneider, K. *Prototypes as Assets, not Toys. Why and How to Extract Knowledge from Prototypes*. in *18th International Conference on Software Engineering (ICSE-18)*. 1996. Berlin, Germany.
[22] Differding, C., B. Hoisl, and C.M. Lott. *Technology package for the goal question metric paradigm*. University of Kaiserslautern, 1996.