

Communication is the key – Support Durable Knowledge Sharing in Software Engineering by Microblogging

Wolfgang Reinhardt
University of Paderborn, Institute for Computer Science
wolle@uni-paderborn.de

Abstract: Communication is undoubtedly one of the key elements of successful software development. Especially in larger groups communication is the critical point in gathering and forming relevant information, share knowledge and create functioning products. Some studies stressed out the fact that informal, *ad hoc* communication take up a significant part of the developers working time. Nonetheless the support of inter-project and inter-organisational communication seems to play a minor part in the development of IDEs and software development platforms. In this paper we discuss communication and knowledge sharing in software engineering and introduce an approach to support social software engineering by microblogging. This approach is to be studied in future projects.

1 Introduction

People and their interactions are the new topic of interest in the past few years. Software developers are embedded in a social framework of colleagues, projects and organisations and produce many types of artefacts for a broad spectrum of people. Beside of the various needed working appliances, communication is undoubtedly a critical point in the process of software engineering especially those in groups. Communication plays an important role in gathering and forming of all relevant information, dependences and users needs and wishes (Cherry and Robillard, 2004; Seaman, 1996). Several studies stressed the fact, that informal *ad hoc* communications are a very important part of the overall communication in a software engineering team. These *ad hoc* activities are defined as interactions forming a logical communicative unit containing one or more sequences with obvious internal continuousness, while remaining unattached from the ambient process (Cherry and Robillard, 2008).

Even though these ad-hoc conversations represent an important part of the daily work of a developer they are rarely or even not supported by the instruments available for supporting software development processes. On the other side, spoken communication or those via other chat tools are mostly fleeting and therefore not useful for a durable process of knowledge sharing and further development. Within the last two years a novel mode of communication emerged with microblogging, which could also be advantageous for the use in distributed and co-located software teams as it can be saved persistently and is further kept out of restrictions of formal communication. In the meantime, there are multiple open source microblogging applications available (cf. THWS, 2008), which can be

integrated into own tools and thus may concur to very specific operational areas.

In this paper we give a brief overview of communication in software engineering, the difficulties of knowledge management and sharing in SE teams and introduce an approach of integrating microblogging to the main working appliance of software developers: the IDE.

2 Conceptual Background

In this section we provide a conceptual framework of key factors, definitions and constructs that are the basis for our paper. The elements of this framework are knowledge sharing and communication in software engineering projects, social software engineering and microblogging.

2.1 Knowledge Sharing and Artefacts

Nonaka and Takeushi (1995) propose a knowledge-sharing model in which the knowledge-sharing process is a loop of socialisation, externalisation, internalisation and combination of knowledge. This so-called *tacit-explicit model* or *SECI-model* is shown in figure 1.

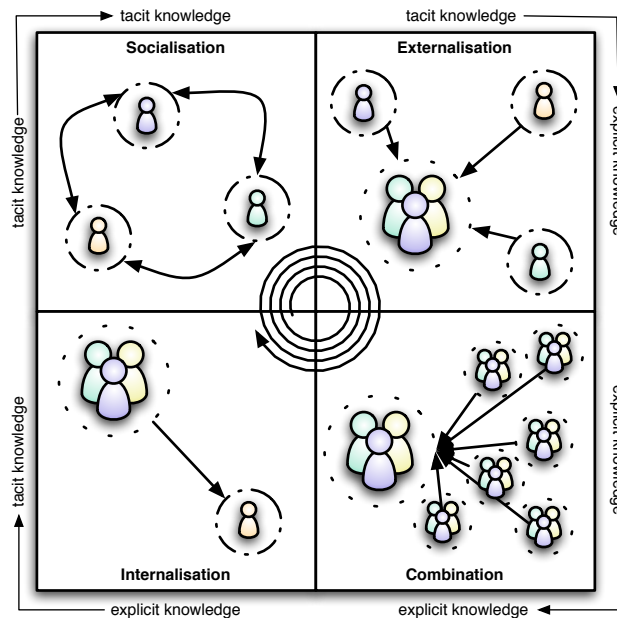


Figure 1: Knowledge-Sharing Model. Adapted from Nonaka and Takeushi (1995)

According to Nonaka and Takeuchi (1995) the continuous cycle starts with the socialisation, where tacit knowledge is transferred between individuals. The next step of the cycle is externalisation, in which the individuals make their tacit knowledge explicit to a group or organisation by means of creating an artefact. By combination explicit knowledge is then transferred into more matured or sophisticated knowledge. Lastly this explicit knowledge is absorbed by individuals, combined with their own knowledge and experience and transferred into new tacit knowledge. The SECI-model fosters learning and knowledge gain by the bias of continuous externalisation and internalisation of knowledge artefacts. Keil-Slawik (1992) describes artefacts as external memories: *owing to their physical nature, artifacts function as external memory, thus facilitating communication and learning* (p. 179).

However, the process of software engineering must not be regarded as a solely technical process of creating artefacts. A working product that fulfills the requirements of all stakeholders is the main goal of each software development project. But it is also associated with the application domain and the acquisition of skills and knowledge by all concerned parties (see Magenheim and Schulte, 2006). According to Engbring (2003) the creation of artefacts depends on cognitive and social aspects. Artefacts are regulating social aspects and lead to new cognitive abilities. So the experience with existing artefacts leads to the creation of socio-facts (see figure 2). Socio-facts are structural elements that describe laws, rules and conventions. On the other hand, the resulting artefacts of a software development project generate new individual and organisational knowledge and necessitate the reflection and acquisition of new skills and socio-facts.

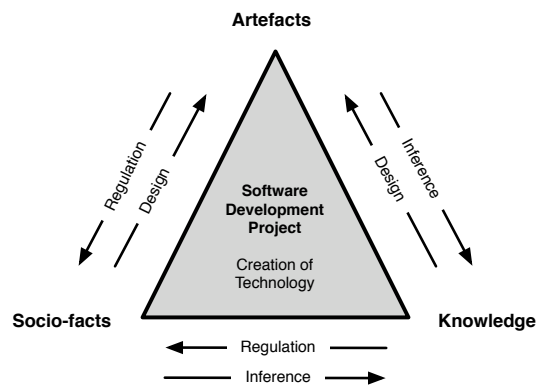


Figure 2: Technology Triangle. See Krohn (1992) and Engbring (2003).

With the SECI model and the technology triangle we demonstrated the artefact-based transformation from tacit to explicit knowledge and its sharing in groups. The next section describes the importance of communication in software engineering.

2.2 Communication in Software Engineering

Communication is the sine qua non of successful software engineering. It is an obvious way for project members to coordinate work in software development environments. Kraut and Steerer (1995) define communication in the context of software engineering: *In software development, communication means that different people are working on a common project agree to a common definition of what they are building, share information and mesh their activities.*

Each individual in a project acts within several domains. The individual belongs to one or more projects that take place in an organisational framework. The organisation itself is placed within the outer-organisational world and interacts with other organisations and individuals. Communication is very important for distributed teams but also for co-located, inter-organisational relationships between different projects and as a strategy to manage dependencies between them (Malone and Crowston, 1994).

Malone and Crowston (1994) define communication as an *activity that is needed to manage dependencies* between actors in the software development process. Communication is a mediating factor that effects both coordination and control activities of a software project. Without (good) communication neither the one nor the other work out as expected. Both Herbsleb and Mockus (2003) and Kraut and Streeter (1995) as well as Perry et al. (1994) divide communication into formal and informal communication. Informal communication is considered as explicit communication via diverse communication channels such as telephone, video, audio conference, voice mail, e-mail or other verbal conversations. Formal conversation refers to explicit communication such as written specification documents, reports, protocols, status meetings or source code (cf. table 1).

Table 1: Communication Types and Techniques (cf. Pikkarainen et al., 2008)

Communication types	Communication techniques
Informal	<ul style="list-style-type: none">* Face-to-Face discussions in co-located or distributed teams* Informal discussions by use of sundry communication channels* Any kind of <i>ad hoc</i> communication
Formal	<ul style="list-style-type: none">* Group, steering group and milestone meetings* Status meetings at which the personnel presents the project results* Formal meetings by use of many communication channels* Formal Documentation, e.g. specification documents, reports or meeting minutes* Source Code

Even though formal communication is an essential part of each software development project, informal communication activities seems to make the difference between *just working* and *great experience* projects. According to Perry et al. (1994) informal communication take up an average of 75 minutes per day for a developer and Seaman (1996) support the need for informal communication *if developers are to carry out their tasks*

adequately.

Robillard and Robillard (2000) point out in their study that collaborative *ad hoc* activities can take up to 41% of the developers time. These *ad hoc* activities are defined as interactions forming a logical communicative unit containing one or more sequences with obvious internal continuousness, while remaining unattached from the ambient process (Cherry and Robillard, 2008). *Ad hoc* communications arises when one team member spontaneously interrupts a team-mate to ask him a question or to provide unsolicited information (p. 495).

Usually, regular face-to-face communication is considered one of the best ways to spread knowledge and build trust in a team. For co-located teams productivity seems to rise, if working in the same room (Espinosa and Carmel, 2003). This is because the co-location fosters collaboration by allowing continuous interactive communication. Software quality rises when two team members team up and collaborate in the agile software practice of pair programming (cf. Reinhardt, 2006). New communication channels in software engineering should try to impose the feeling of a common working room. This impression bolsters the nonchalant, formal and informal use of the communication channel. Herbsleb and Mockus (2003) call this effect the *virtual 30 meters*.

2.3 Microblogging

Microblogging is the latest form of blogging where messages are typically not longer than 140 characters. Templeton (2008) defines microblogging as *a small scale form of blogging, generally made up of short, succinct messages, used by both consumers and businesses to share news, post status updates and carry on conversations*. These micro-messages can be restricted to a certain number of individuals, sent exclusively to a specific contact, or made available to the World Wide Web. Microblogging has impressively become more and more popular in the last two years, and Twitter¹ is probably the most well-known microblogging platform currently available on the web, when compared with other microblogging tools, such as Plurk², Jaiku³ and Pownce⁴ or the Open Source Tool Identica⁵ (Java et al., 2007). While regular weblogs are mainly used for writing short essays, knowledge saving and discourse, microblogging is proving extremely useful for the fast exchanges of thoughts, ideas and information sharing (Ebner and Schiefner, 2008). With the Open Source application Laconica⁶, there is an alternative to Twitter that is usable on own servers and in own educational and professional settings. In the last year we have witnessed the use of such approach as a powerful component of one's networking activity, and more importantly seems to become a relevant part of one's informal learning.

Another interesting part of microblogging is the possibility of hashtagging one's micro-

¹<http://twitter.com>

²<http://www.plurk.com>

³<http://www.jaiku.com>

⁴<http://pownce.com>

⁵<http://identi.ca>

⁶<http://laconica.ca>

posts. The use of hashtags is extremely useful when sharing and contributing to a specific topic, event or project. Hashtags are a simple way of grouping messages together with a # sign followed by a tag. With the aid of this hashtag all messages with the same hashtag can be aggregated and regarded in a larger context.

2.4 Social Software Engineering

The term *Social Software Engineering* can be read by two different means, which cover different fields of software development. If we emphasize the first two words, thus we have [Social Software] Engineering – the process of drafting and creating social software applications. But if we particularly have a look at the last two words Social [Software Engineering], then we talk about social, non-technical aspects of the software development process.

[Social Software] Engineering characterizes the implementation of so called Social Software. The term does not focus on the engineering process as such, but the developed social software. Thus questions like *which types of social software do exist and how do the engineering process of them differ?* are in the focus of research. Furthermore [Social Software] Engineering deals with the problem how social software can be integrated in the IT landscape in enterprises (Enterprise 2.0) and how introductory processes and the management of social software distinguishes from those of non-social software.

Social [Software Engineering] mainly considers social, non-technical aspects of developing software. At this point we need to ask which parts of the everyday work of a software engineer can be described as social relevant. Communication, collaboration and information exchange are in centre of Social [Software Engineering] research – typical results are visualisations of communication flows and social networks. On the other hand motivational aspects and incentives belong to the second understand of the term.

3 Tools to support the Software Development Process

The everyday work of a software engineer is nowadays shaped by a multitude of instruments. The most principle one is the integrated development environment (IDE), where-with the most important artefacts – the source code – are created. An IDE is a piece of software that provides comprehensive functionalities to support software developers. An IDE usually consists of a source code editor, an integrated compiler, automatic build tools and a code debugger. IDEs are designed to maximize the productivity of a developer, because of the highly integrated build tools. Because of its complexity it takes long to understand all facets of an IDE and the high productivity only comes after a long time of working with it. IDEs are strongly put in the centre of focus in regards of process and working tools optimization during the last years, so that they already contain important features of software quality assurance as well as of monitoring. Besides the IDE the developer also operates with a word processor composing documents, reports and meeting

minutes. Additionally, he uses a browser for information searches as well as for use of web-based supporting instruments such as wikis or issue trackers. Furthermore, he acts with an e-mail-client and various instruments of digital communication. According to the project type and structure and the organisational size the multitude of instruments often is extended by applications for software configuration management (SCM) and application lifecycle management (ALM).

As Pressman (2004) states SCM is a *set of activities designed to control change by identifying the work products that are likely to change, establishing relationships among them, defining mechanisms for managing different versions of these work products, controlling the changes imposed, and auditing and reporting on the changes made*. So SCM is the task of tracking and controlling all changes in a software product. Among the practices of SCM are version control and the definition of milestones. ALM is the chaining of business management to software engineering enabled by tools that facilitate and integrate requirements management, architecture, coding, testing, tracking, and release management. (deJong, 2008)

None of the known instruments supports the process of social [software engineering] adequately. They all concentrate strongly on the procedural and organisational assistance of the software development process and provide support to the developer primarily in formal communication, the processing of tasks and the solution of problems. The support of process-oriented interactions between the team members as well as the technical support of informal communication does not happen in most cases. From our point of view, both platforms for SCM as well as for ALM applications should become more social applications and further enhance the interpersonal communication, ad hoc chats and the exchange of experiences. For us microblogging is a slight approach to realise a first social feature (the fast informal broadcast communication). In a consequent development of a social software configuration management platform (SSCMP) further efficient phenomenon of the web 2.0 should be integrated including expert finding, tagging, social bookmarking, etc.

4 Support communication of software engineering teams by microblogging

As mentioned earlier *ad hoc* communication constitutes a main part of the daily work of a software developer. Common organisations run multiple projects with different tasks at the same time, which interface each other. To strengthen the inter- and intra-project-communication, we propose to implement microblogging within software engineering. Perry et al. (1994) and Cherry and Robillard (2004) point out that many intra-project issues are solved by spontaneous *ad hoc* communication of the project members. This problem solving potential also exists in inter-project settings. When members of different projects meet, they discuss problems, solutions and best practices from their projects. This information reaches a wider organisational public and thus contribute to knowledge acquisition and learning. Nevertheless these spoken communications are problematical, since

the information exchanged are fleeting. As long as the communication is not recorded, transcribed or save in any other kind of persistent artefact, the information cannot be used in other setting, from other individuals or groups.

Many software development teams use instant messengers to quickly exchange thoughts, information and shouts for help. These messages as well are not usable during a downstream information search. As long as the content of the chat is not externalised in some way⁷ and made explicit, the knowledge is unusable for the working and learning process. Individual and organisational knowledge development is very tight coupled to this cycle of tacit-to-explicit-to-tacit knowledge sharing. The SECI model of Nonaka and Takeushi (1995) as well as the technology triangle (Krohn, 1992 and Engbring, 2003) show that knowledge acquisition is mainly based on externalised artefacts. Only these seizable information objects allow the individual and organisational knowledge development to take place (cf. Keil-Slawik, 1992). Each microblogging service provider saves all the sent messages persistently. Each message stays accessible until the sender decides to delete it⁸ and thus can be accessed from any person in the senders network. If we assume that every developer is mutual connected to each other developer in an organisation, each of the messages can durable be searched for hints, solutions or experts.

4.1 A Prototype for Microblogging from the IDE

Eclipse⁹ is one of the frequently used integrated development environments (IDE) in contemporary software development projects. Because of this, and the ease of extending the platform with own plugins via the plug-in architecture we choose Eclipse for developing a microblogging plugin. Figure 3 shows the first prototype of an Eclipse Twitter client.

The prototype has been created based on the Eclipse Rich Client Platform (RCP) and integrates itself seamless in the user interface of the IDE. The credentials for the Twitter account are once saved in the Eclipse system preferences for each developer. The plugin creates a specific view in Eclipse with which messages via Twitter can be send. The plugin uses the API provided by Twitter to send messages. Because of the seamless integration of the IDE the developer does not need to leave his working environment for sending messages but is able to directly continue on his coding work. Due to the flexible use of hashtags in Twitter messages the developer can easily change the context (and thus the recipients) of his message.

⁷There are many ways to externalise the content of such a quick chat: one of the concerned team members could write an e-mail, a report, or just a source code comment.

⁸Maybe it is a good idea to store all messages redundantly to counter vandalism or accidental deletes. Of course this one is a very critical point due to data privacy and personality rights.

⁹<http://www.eclipse.org>

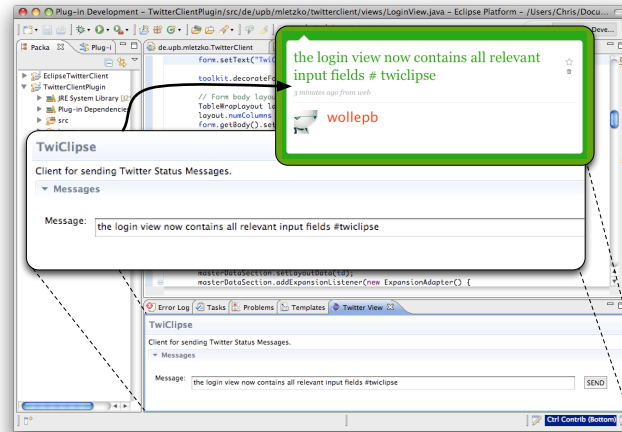


Figure 3: Twitter Client in Eclipse

4.2 What's the added value of this approach?

Microblogging does not restrict the developer to send normalised, formal status messages (similar to log messages at the check-in into a repository) or completely informal chats like in instant messaging. With microblogging the developer can talk about successful implementations, problems in developing a specific method or just shout for help. We do not propose to replace personal spoken communication during coffee breaks or a pair programming session, but however only to facilitate the anyway happening communication with another communication channel.

For one thing, the presented prototype meets the requirement of information externalisation as artefacts – as a kind of external memory – to foster the knowledge development of individuals, groups and organisations. Due to the open design of microblogging and the many possibilities of hashtagging the exchanged information can be used both in intra-project and inter-project settings (see figure 4).

For another thing, the messages sent are durable accessible by the persistent storage of the service provider and can be integrated into existing SCM or ALM applications via hashtag aggregation. Contingent on the possibility of informal communication with other team members the microblogging channel create the impression of a common room and therefore with the effect of the virtual 30 meters. The integration into the main working instrument of a developer has exposed to be a good choice since the no longer necessary tool shift is assessed as good and leads to a high readiness of use among developers.

With a possible extension of the prototype the opportunity arises to associate multiple artefacts (e.g. source code files and Twitter messages) with the belonging persons. In the resulting artefact-person-network it is easy to discover connections between artefacts and persons that are not obvious without this visualisation. This kind of network can then be

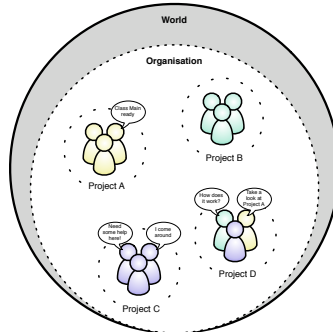


Figure 4: Intra- and Inter-Project Communication with Microblogging

used to find experts for specific topics, classes or programming languages or even for the introduction of new team members.

5 Conclusion and Outlook

In this paper we have seen the importance of knowledge sharing and communication in software engineering projects. We have recognized that both personal and organisational knowledge development is tightly coupled to the externalisation of communication. Only the durable supply of information as artefacts makes it possible to support inter- and intra-project communication and the sustainable transformation from tacit to explicit knowledge in various domains. In the course of this it is more or less non-relevant whether the artefacts results from formal or informal communication. The minutes of a status meeting can be of same importance for the development of a team as an informal shout for help via a microblogging message.

The presented Eclipse-based prototype will be used in a computer science class in the upcoming summer term. In an ethnographic study the use and acceptance of the prototype will be studied and presented in an upcoming paper. In the mid term the prototype shall communicate with a local Laconica server rather than with Twitter and thus be connected with other running projects in the area of social software engineering. In the long term the presented approach will be integrated into an emergent platform for social software engineering (SSCMP) that supplies further features of social software.

References

- [CR04] Sébastien Cherry and Pierre N. Robillard. Communication Problems in Global Software Development: Spotlight on a New Field of Investigation. In *Third International Workshop on Global Software Development*. ICSE, 2004.

- [CR08] Sébastien Cherry and Pierre N. Robillard. The social side of software engineering—A real ad hoc collaboration network. *International Journal of Human-Computer Studies*, 66(7):495 – 505, 2008. Collaborative and social aspects of software development.
- [deJ08] Jennifer deJong. Mea culpa, ALM toolmakers say. <http://www.sdtimes.com/content/article.aspx?ArticleID=31952>, April 2008.
- [EC03] J. Alberto Espinosa and Erran Carmel. The impact of time separation on coordination in global software teams: a conceptual foundation. *Software Process: Improvement and Practice*, 8, 2003.
- [Eng03] Dieter Engbring. *Informatik im Herstellungs- und Nutzungskontext*. PhD thesis, University of Paderborn, 2003.
- [ES08] Martin Ebner and Mandy Schiefner. Microblogging - more than fun? In *Proceedings of the IADIS Mobile Learning Conference*, pages 155–159, 2008.
- [GC05] Tim Goles and Wynne W. Chin. Information systems outsourcing relationship factors: detailed conceptualization and initial evidence. *SIGMIS Database*, 36(4):47–67, 2005.
- [GR08] Judith Good and Pablo Romero. Collaborative and social aspects of software development. *International Journal of Human-Computer Studies*, 66(7):481 – 483, 2008. Collaborative and social aspects of software development.
- [HM03] J.D. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *Software Engineering, IEEE Transactions on*, 29(6):481–494, June 2003.
- [JMT05] Michael John, Frank Maurer, and Bjørnar Tessem. Human and social factors of software engineering: workshop summary. *SIGSOFT Softw. Eng. Notes*, 30(4):1–6, 2005.
- [JSFT07] Akshay Java, Xiaodan Song, Tim Finin, and Belle Tseng. Why We Twitter: Understanding Microblogging Usage and Communities. In *Proceedings of the Joint 9th WEBKDD and 1st SNA-KDD Workshop 2007*, August 2007.
- [Kro92] Wolfgang Krohn. *Technik-Kultur-Arbeit*, chapter Zum historischen Verständnis der Technik, pages 27–34. 1992.
- [KS92] Reinhard Keil-Slawik. *Software Development and Reality Construction*, chapter Artifacts in Software Design, pages 168–188. Springer, 1992.
- [KS95] Robert E. Kraut and Lynn A. Streeter. Coordination in software development. *Commun. ACM*, 38(3):69–81, 1995.
- [MC94] Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Comput. Surv.*, 26(1):87–119, 1994.

- [MS06] Johannes Magenheimer and Carsten Schulte. Social, ethical and technical issues in formatics - An integrated approach. *Education and Information Technologies*, 11(3-4):319–339, 2006.
- [NT95] Ikujiro Nonaka and Hirotaka Takeuchi. *The Knowledge Creating Company: How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press, 1995.
- [PHS⁺08] M. Pikkarainen, J. Haikara, O. Salo, P. Abrahamsson, and J. Still. The impact of agile practices on communication in software development. *Empirical Softw. Engg.*, 13(3):303–337, 2008.
- [Pre04] Roger S. Pressman. *Software engineering : a practitioner's approach*. McGraw Hill Higher Education, 6 edition, 2004.
- [PSV94] Dewayne E. Perry, Nancy Staudenmayer, and Lawrence G. Votta. People, Organizations, and Process Improvement. *IEEE Softw.*, 11(4):36–45, 1994.
- [Rei06] Wolfgang Reinhardt. Einfluss agiler Softwareentwicklung auf die Kompetenzentwicklung in der universitären Informatikausbildung. Analyse und Bewertung empirischer Studien zum Pair Programming. Master's thesis, University of Paderborn, 2006.
- [RR00] Pierre N. Robillard and Martin P. Robillard. Types of collaborative work in software engineering. *J. Syst. Softw.*, 53(3):219–224, 2000.
- [Sea96] Carolyn B. Seaman. *Organizational Issues in Software Development: An Empirical Study of Communication*. PhD thesis, University of Maryland, 1996.
- [SMB⁺04] Forrest Shull, Manoel G. Mendonça, Victor Basili, Jeffrey Carver, José C. Maldonado, Sandra Fabbri, Guilherme Horta Travassos, and Maria Cristina Ferreira. Knowledge-Sharing Issues in Experimental Software Engineering. *Empirical Software Engineering*, 9(1-2):111–137, March 2004.
- [SN04] John A. Scott and David Nisse. *Guide to Software Engineering Body of Knowledge*, chapter 7: Software configuration management. IEEE Computer Society, 2004.
- [SR05] Helen Sharp and Hugh Robinson. Some social factors of software engineering: the maverick, community and technical practices. In *HSSE '05: Proceedings of the 2005 workshop on Human and social factors of software engineering*, pages 1–6, New York, NY, USA, 2005. ACM.
- [Tem08] Mike Templeton. Microblogging Defined. <http://microblink.com/2008/11/11/microblogging-defined/> (02.12.2008), November 2008.
- [THW08] THWS. The Twitter-clone/twitter-like sites collection. <http://www.thws.cn/articles/twitter-clones.html> (02.12.2008), 2008.