Benefits and Challenges of Using Collaborative Development Environments with Social Software in Higher Computer Science Education

Daniel Kadenbach and Carsten Kleiner

University of Applied Sciences and Arts Hannover, Germany
{daniel.kadenbach, carsten.kleiner}@fh-hannover.de

Abstract. This paper addresses the question how to optimally support projects of students and employees of a higher education institution of computer science by means of a special software environment. At first the motivation to introduce such a supportive system is examined by describing the current situation in the authors' department of computer science, which is typical for many colleges and universities. On the one hand problems are pointed out, which hamper students and employees in their project work, on the other hand the additional possibilities of a supportive system, which far exceed the ones of a traditional approach, are drafted. The paper shows how a mutual value for students and employees can be generated from the projects by using social software. After the requirements are described we suggest an architecture for such a supportive system and finally the challenges for the implementation and application, which determine the success or failure of the system, are discussed.

1 Introduction

In times with an ever increasing demand for flexibility of employees in the field of computer science, in which agile (virtual) teams have to respond to fast changing requirements and to overcome distances of time and space, a sophisticated and comprehensive collaboration support for projects becomes crucial.

This paper therefore looks into the questions of how traditional and virtual teams in software-development projects can be optimally supported by collaborative development environments (CDEs[1]) and how an application of them could improve the quality of education of students and work of employees in colleges and universities in the field of software development. Besides conventional CDEs, which mainly focus on technical aspects of software development (like providing a project overview, team management, version control systems such as Subversion, bug-, feature- and task-trackers and a release management), the look is broadened to cover also social aspects of supporting functions. The strengths and risks of a solution are analyzed, in which findings from social computing and from online communities are used for the design

[1] CDEs are also called forges, for examples see SourceForge.net (http://sourceforge.net), GForge (http://gforge.org) or LibreSource (http://dev.libresource.org).

A.A. Ozok and P. Zaphiris (Eds.): Online Communities, LNCS 5621, pp. 479–487, 2009.

of a CDE, because one aspect which has been often neglected so far is how the establishment of good teamwork can be supported with the help of social components. We suggest to reach an additional value through a stronger integration of social tools to give an answer to questions like:

- How to support the forming of teams?
- How to increase trust and identification within a team and to support the forming of team spirit and a constantly high level of motivation?
- How to generate self-dynamics and more creativity, like it can be seen in many social computing applications?
- And finally how to generate a lasting value for the general public from regular projects and simultaneously minimize the risk of producing data waste?

In the main part the requirements for a CDE at the authors' department of computer science are analyzed and components are discussed, which are used to realize the system and therefore fulfill these requirements. An overall system is drafted, which provides a far greater benefit through the use of synergy effects, as the individually components would do. Therefore an integration of these components among each other as well as the addition of some overall functions (like a comprehensive search facility over all components, tagging of all information, fast and simple ways to edit all data, easy and efficient handling) are of great importance. It is exposed how critical for the success of such a system are intuitive usability and self-evident benefits seen by its users and also which factors could lead to a refusal of the system by its users. Finally the drafted model of a CDE with social functions is discussed summing everything up. The paper closes with a conclusion and discussion of open points.

2 Related Work

The individual components of the drafted system like portals, wikis, groupware, blogs, trackers, version control systems, discussion forums are well-known and have mostly been investigated in many other contexts (for example see [1] for a general introduction to practical development environments). This work particularly focuses on the concurrence of the components in a holistic context with the aim to support not only projects for themselves but to establish a self-supporting community. This system therefore depends on mechanisms for team building, team spirit, motivation, awareness and user acceptance which also have been investigated in different contexts. Also the aspects of supporting learning with collaborative environments have been discussed [2] [3] [4].

Another important point is the management of knowledge, especially the benefits of cross-project knowledge collaboration, which is described in [5]. For future versions of the drafted system inter-organizational collaboration will also be of great interest [6].

The work from Karen L. Reid and Gregory V. Wilson concerning their DrProject web-based software project management portal has much in common with our work, because their system has similar requirements, also aims to meet educational needs and offers similar functionality [7] [8]. But our work primarily tries to focus more on the social aspects of the system to enable the growth and consolidation of a vital

community in which project participants help and profit from each other beyond project boundaries and to ensure a sustainable value of projects carried out. Also our work chooses a different way in the implementation by combining and integrating multiple F/OSS components to built the system (see Section 6.1 for further details).

3 Method

To develop a system for the support of software development projects in our department a user-centric method was chosen, so the future users were involved in the whole process. Initially an analysis of the current situation was carried out, which motivated the development and at which projects and old project environments were investigated and project participants were questioned. Afterwards the requirements for the system were compiled by additionally using results of a poll carried out with the future users and general considerations and findings of the CSCW research field. With the raised requirements existing software systems and components were evaluated with the assistance of future users and finally a protopical system was developed. It was used right from the start to support real projects to refine the system with the gained insights and user experiences.

4 Initial Situation

Before describing the requirements it seems to be sensible to picture the initial situation and the problems which motivated the implementation of a supportive system for projects in the department of computer science at our institution. Every semester and beyond many projects are carried out by students, employees and faculty partly in cooperation with externals companies and greatly differing in size and subject. Some of the projects are solely used for educational purposes, some for productive use and some for scientific research. Most of the projects are directly or indirectly located in the field of software-development. So up to now the core requirements for a project environment usually consisted of some of the following points:

- Mailing lists supporting communication of the project group,
- tools for shared version control of files like Subversion,
- shared folders for documents and files (but hitherto without a document management system),
- a wiki for online-documentation of the project,
- tracker to document and handle defects and feature requests and
- a project server to deploy and test the developed applications.

The usual way for carrying out such projects has been to give projects a mailing list and a project server on request. On this server the project participants then could install all required components and services by themselves. Smaller projects by students however were more or less on their own. Whereas they typically have a smaller need for infrastructure support, they also should not be neglected since their number is large and thus there is a great potential for improved results.

It is obvious that through the self-administration of every project infrastructure there is not only an avoidable overload of administrative tasks and required hardware (because nearly no project is able to reach the performance limit of its project server on its own) but also the projects are mostly isolated from their environment in this way. One of the most fundamental shortcomings of this approach, which has been identified in the analysis of the initial situation, is mainly caused because of this isolation: In almost none of the carried out projects the full potential of the project achievements was used onwards.

The projects with all their achievements (gathered knowledge and experiences, source-code, applications, etc.) were mostly stored in such a scattered way that shortly after their end everything faded into void and no one could benefit from them anymore. This situation is especially bad because on the one hand the projects often achieve remarkable results which deserve to be appreciated and reused in later projects. On the other hand students, employees and the faculty of the department are a highly motivated and well-versed community, which could mutually benefit from projects carried out, if their results were accessible by everybody in the institution.

5 Requirements

Through the analysis of the initial situation, the survey and interviews with students and employees as well as a prototypical implementation of a supportive system directly evaluated by persons from the target groups with real projects, the following requirements were identified:

- Project Portal. First of all it is important to establish some sort of centralized portal for all projects and all related data or alternatively multiple portals for different kinds of projects like solely educational, practical or scientific ones. The portal should be the central point where to browse through all carried out and currently running projects and search through their contents. It therefore would guarantee that the projects are not isolated from each other, that they will not silently fade away into the void after being finished and that they are accessible by the community. Also all supportive functions shall be usable or at least referenced in the project portal and shall be activateable through the portal without the need of any administrative action. A project portal and therefore the presentation of the projects could also encourage project participants to create better projects, because their work will be visible for the community and so there is more transparency, pressure and recognition.
- Version Control System. Since crucial in software development the system should support at least one version control system like Subversion, also without administrative action. To support the awareness inside the projects, the users should be automatically notified of changes after a commit with sufficient context information, so they stay up to date and are motivated through the visible project progress. Additionally there should be a comfortable web-interface to the version control system, on the one hand so that users can easily browse the contents and compare different versions of files, on the other hand this also enables people outside the project to learn from the code and other artifacts, especially if the content is also searchable. As a side effect there would probably be a higher motivation to write

readable code, because it can be easily seen be everyone in the community. Ideally the system should provide the possibility for users to write comments to the files they browse, so they can give feedback. Also at this point source-code analysis tools could be integrated into the system, which could automatically give feedback about the quality and possible problems in the code.

- Shared File-Access. Project members should have a simple way of sharing unversioned files also without the burdens of a shared versioning system, so external documents or large binary files can be accessed easily.
- Project Representation and Documentation Support. The system should enable the projects to create a homepage for their representation, either by hand or preferably by using preinstalled CMS, and also should have the possibility to create wikis and blogs to support their documentation.
- Communication Support. The system should be able to automatically create a mailing-list for the participants of a project and to create discussion forums for a project on demand.
- Feature and Bug Tracker. For the gathering of needed functionality and the documentation and remedy of bugs feature and bug tracker shall be available for the projects. Ideally the feature and bug entries are referenceable from other components of the system, like the version control system or wikis.
- Knowledge Management Support. In addition to wikis and blogs, knowledge management tools like document management systems and FAQ-databases should be available, so that project members can document solutions for the problems they encounter in their project work and share them with the community.
- Project Internal Organization Support. Functions of groupware applications like shared task-, time-, resource-, calendar- and contacts-management should be supported on demand.
- Browseable, Searchable, Commentable and Tagable Content. Ideally every content of the system should be browseable, searchable, commentable and tagable. This shall ensure that information can be categorized, found and is generally accessible, that feedback can be given and discussed to improve the content.
- Security and Privacy Protection. As the social functions would expose a lot of data about the project participants, it has to be ensured, that their privacy is protected. Either the system could only be accessible for community members or the member names could be made anonymous for viewers which are not part of the community.
- Single-Sign-On. A central management of user accounts and rights is needed, so that a user only has to log in to the system once, even if it consists of multiple components.
- Tailoring of the Functions. Functions of the system which are not needed for a particular project should be blinded out, so that they do not hinder or confuse the users in their work.
- Simplicity. The system should be as simple as possible, so that it can be started up, administered and used with minimal effort.
- Ease of Use. The system and all of its components should be easy to use. This includes a user-friendly intuitive user interface and a little learning curve for the usage of the system. Ideally the system should consist of components which are already known to the users.

- Extensibility. Because of the diversity of projects it is essential for the system that it could be extended with new components easily and that existing components can be updated likewise. Otherwise it would be a matter of months and the system would be stale and possibly insecure.
- License. License costs for proprietary software were no option for this project. So the software system and/or used components should be free and open source software (F/OSS). This also improves the chances to extend and upgrade the software.

The carried out survey with students as the main target group of the system also showed that 100% of the students which answered the poll wanted a supportive system and that more than 80% were interested in the benefits of social functions even if they had to invest time and work to support the community.

6 Suggested Architecture

Generally there are three possible approaches to implement such a supportive system: (1) use a complete existing software-system and adapt it to your needs, (2) completely implement the system yourself or (3) implement a framework which can include existing components. All three approaches were investigated carefully. Using a complete existing software-system at first looked like the approach which requires a minimal effort. But at the time of the investigation no F/OSS solution could be found which was able to comply with the requirements or looked like it could be modified to do so with moderate effort. Approach two will probably be preferred by most developers, because it would offer the most freedom in designing and implementing the system from scratch with respect to the requirements. But this approach was also not taken because firstly the drafted system is fairly complex as can be seen from its requirements: even the design and implementation of some of its components would require a great effort and the whole system would not be realizable in the given timeframe with the given resources. Secondly and actually more importantly, even if it would have been possible to develop the system from scratch, it would probably not be possible to keep every component up to date over time and add adequate new components. The effort for such customizations and extensions would be more than the department could accomplish because of its small size and therefore limited resources.

The last approach - implementing a framework which can include existing components - which was finally chosen, does have a few significant advantages: There already exist F/OSS for nearly every component the system needs, many users are accustomed to use them, the components are developed and updated independently and it would not impose a great effort to change, remove or add components to the system, after the framework has been built. But also some challenges were identified which should not be underestimated: First of all the integration of the components into a coherent system may require changes or adjustments in all of the components for overall functions if they do not already contain standardized interfaces. For example for the authentication to realize a single-sign-on system, for doing a search over the contents of all components or for awareness support (so that a team member can automatically be informed if there is a change in a component) every component must support the corresponding functionality of the integrated system or must be modified

respectively. Furthermore the value of the entire system depends also on the integration of the components among each other to realize synergetic effects. For example it could make the system more efficient and transparent if the bug-tracking system is able to gather information from commits into the source-code repository, which contain IDs of bugs in their comments, or if it is possible make a direct link in a discussion forum entry to a file in the repository.

Interestingly, a leading CDE-hosting site SourceForge.net is also moving towards the last approach, as it is giving its users the option to automatically install different F/OSS applications to support their projects which is called "hosted apps" [9].

6.1 Overall Architecture

Because nearly all of the required components for the system like wikis, CMSs, blogs, bug-tracker, discussion forums and so on are available as F/OSS implemented with the scripting language PHP an apache web-server was chosen to be the core of the system. For the version control system Subversion was chosen for the prototype but it is obvious how to support different version control systems in the future. Subversion can be configured so that it can also be accessed through apache. This has the advantage that the system needs fewer ports and that security as well as authentication mechanisms of the apache can be used for all the components likewise. So all webpages of the components and the access to the subversion system can be secured with HTTPS. Furthermore the system has to be able to send and receive mails, must provide databases for the components, ways for file access for the users, an administration interface and many other functions. As seen drafted in Fig. 1 it therefore is a complex server system.

6.2 Virtualization

To respect the requirement of an easy installation with this complex system and to reach a greater flexibility of the system it is convenient to use a virtualization approach. Therefore the prototypical implementation which was used for the test phases was a virtualized debian linux system, on which all required software is installed and configured together with the framework for the components. In this way the whole

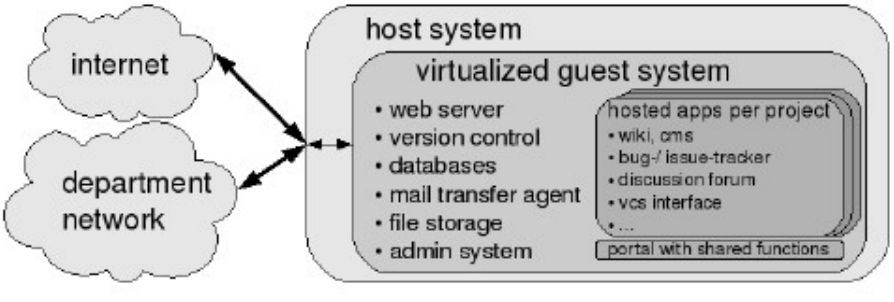

Fig. 1. Overview of the Architecture of the Prototypical Implementation

complex system can be installed with minimal effort and runs on every system which supports software for the chosen virtualization method. This also makes it possible to easily migrate the system from one server to another, enables easy system-backups and, in the case of a compromised system, a clean system can be reinstalled fast and without trouble.

Another possible advantage of the virtualization is that it would enable the system to easily spawn other virtualized instances of the base installation, which could provide projects with the access to a virtualized project server instance, which provides full administrative control and at the same time does not impose a security risk for the whole system. On this system the projects could test their developed applications. This could also significantly reduce hardware requirements of a department, because many virtualized project server instances could be run on the same physical machine.

6.3 Challenges and Benefits

The greatest challenge of our system does not lie in the technical realization of its several functions, even though there are challenging and interesting aspects in particular in the realization of the overall system functions and the integration of the components among each other. The greatest challenge of such a system is its acceptance by its users. The users can be divided into three groups: end-users of the system, administrators and developers. Because the use of a system which supports projects always is completely voluntary, only functions will be accepted which offer the user an immediate value, which far exceeds the effort to use the system and only then may it be used efficiently. A value can be generated through providing required technical infrastructure, sustainable management of knowledge, improving communication, building a self-supporting community, but also by increasing motivation and fun using the system. With a shared project portal the community shall be strengthened.

7 Conclusion and Future Work

This work does not develop or investigate new components for the support of software development projects in universities, but it tries to draft a way to combine existing proven components into an overall system, which makes it possible to reach an sustainable value for the community through sharing results from projects carried out. Therefore the technical tools for the software-development shall be complemented with social components to improve communication, motivation, personal commitment and the mutual support of project participants.

It seems obvious to prepare students in the IT field for work in virtual teams and the extended possibilities of project support systems like CDEs. But the aim of a CDE with social functions used at the department of computer science is not only to teach the handling of corresponding tools to the students and support them in their project work. The aim is also to increase their motivation, their commitment and generally the value of their projects by making their results, experiences and acquired knowledge accessible to other students and therewith generate a value for the department by using mechanisms of virtual communities. Therefore the system, which is currently at a prototypical development stage (but even though is already used productively), shall be implemented completely so that acceptance of the target groups can be investigated in more detail and further research and reasoning can be undertaken.

Additionally future use cases can be drafted: So it would be a good exercise to let students work in real virtual teams where team members are not able to meet physically, which could be realized if several universities collaborate and let their students work together on projects and communicate virtually through CDEs. This would also strengthen the connections between different institutions and the sharing of knowledge between them. Furthermore in other areas than software-development supportive social community functions for students are imaginable. So the major part of the students in the carried out poll voted for a solution supporting the creation of seminar- and final papers by allowing the community to proofread and comment the works.

Acknowledgements. Daniel Kadenbach wishes to express his gratitude to Dipl. Inf. Casper Sørensen for his valuable suggestions. The authors thank the IT infrastructure staff of the department of computer science of the university of sciences and arts Hannover, especially MSc. Frank Müller, for their help and support, all students and employees of the department which contributed to the system through their ideas, critique, evaluation of the prototype and feedback.

References

1. Doar, M.: Practical Development Environments. O' Reilly & Associates, Inc., Sebastopol (2005)
2. Langton, J.T., Hickey, T.J., Alterman, R.: Integrating tools and resources: a case study in building educational groupware for collaborative programming. J. Comput. Small Coll. 19(5), 140–153 (2004)
3. Giannoukos, I., Lykourentzou, I., Mpardis, G., Nikolopoulos, V., Loumos, V., Kayafas, E.: Collaborative e-learning environments enhanced by wiki technologies. In: PETRA 2008: Proceedings of the 1st international conference on Pervasive Technologies Related to Assistive Environments, pp. 1–5. ACM, New York (2008)
4. Bouillon, P., Krinke, J.: Using eclipse in distant teaching of software engineering. In: eclipse 2004: Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange, pp. 22–26. ACM, New York (2004)
5. Ohira, M., Ohsugi, N., Ohoka, T., ichi Matsumoto, K.: Accelerating cross-project knowledge collaboration using collaborative filtering and social networks. In: MSR 2005: Proceedings of the 2005 international workshop on Mining software repositories, pp. 1–5. ACM, New York (2005)
6. Nuschke, P., Jiang, X.: A framework for inter-organizational collaboration using communication and knowledge management tools. HCI (15), 406–415 (2007)
7. Reid, K.L., Wilson, G.V.: DrProject: a software project management portal to meet educational needs. In: SIGCSE 2007: Proceedings of the 38th SIGCSE technical symposium on Computer science education, pp. 317–321. ACM, New York (2007)
8. Allen, E., Cartwright, R., Reis, C.: Production programming in the classroom. In: SIGCSE 2003: Proceedings of the 34th SIGCSE technical symposium on Computer science education, pp. 89–9U:\BPO\Lncs\5621\Reference\562104933. ACM, New York (2003)
9. SourceForge, Inc., `http://sourceforge.net` (visited, January 2009)
10. GForge Group: GForge Collaborative Development Environment CDE, `http://gforge.org` (visited, January 2009)
11. LibreSource Consortium: LibreSource, `http://dev.libresource.org` (visited, January 2009)