Experience Management Wikis for Reflective Practice in Software Capstone Projects

Eric Ras, Ralf Carbon, Björn Decker, and Jörg Rech

***Abstract*—Software engineering curriculum guidelines state that students should practice methods, techniques, and tools. A capstone project is one possibility to address this aim. A capstone project helps the students to increase their problem solving competencies, improve their social skills (e.g., communication skills), and gather practical experience. A crux of such projects is that students perform "reflective" practice in order to learn from their experiences. The authors believe that experience gathering and reuse are effective techniques to stimulate reflective activities. An adapted free- and open-source Wiki-based system called *software organization platform* (SOP) is used to support students in managing their observations and experiences. The system can be used for experience exchange within the team and for experience reuse in forthcoming projects. The results of a case study show that standard Wiki functions improve communication and information sharing by means of explicit observation and experience documentation. A total of 183 documented observations and experiences at the end of the project provide a measure for the amount of reflection students have had during the capstone project. Still, the advantages of using Wikis will decrease when no technical adaptations of the Wiki to the learning objectives and to the software engineering tasks are made. Limitations of the case study, future evaluation steps, and planned developments of SOP will be provided in this paper.**

***Index Terms*—Capstone project, experience management, knowledge-based systems, open-source software, reflection, software engineering.**

I. Introduction

In almost every software engineering curriculum, a capstone course is a mandatory curriculum component. Educational research in this domain has shown that practicing the learned methods and techniques is essential before the students get involved in industrial software development projects. Shaw *et al.* also stated the importance of practicing what was learned as a core pedagogical principle of software engineering education [1]. Practicing should be performed on a reflective basis since the students must learn to judge about the application of specific methods and techniques, and to evaluate critically the consequences of their actions and decisions. Capstone projects have shown to support self-directed and experiential learning [2], where students reflect and interpret their experiences to build abstractions (e.g., models, principles, strategies, theories, etc.), which are applied and tested in new situations and which provide the foundation for having new experiences.

The Software Engineering Curriculum Guidelines (SE2004) [3] provide learning objectives, sample courses, and key knowledge areas that should be taught to undergraduate students. Two interesting learning outcomes to be highlighted here are: "Show mastery of the software engineering knowledge and skills necessary to begin practice" and "Demonstrate skills such as interpersonal negotiation, effective work habits, leadership, and communication." SE2004 also mandates that students undertake a capstone project to expose the students to the application domain, and that software engineering (SE) should be taught as a problem solving discipline.

However, many capstone projects risk overloading students because they get overwhelmed with so many new topics, because they have to understand the different roles and responsibilities assigned, and because they have to cope with obstacles (e.g., changing software requirements) [4]. Teachers want to provide realistic projects and conflicting situations as they happen in the real world to prepare students for their jobs [5]. In addition, students are supposed to pass through all the technical development phases and perform project and quality management. Umphress *et al.* analyzed 49 capstone projects at the graduate and undergraduate level and stated that most students had difficulties in balancing and estimating the workload during the project, that configuration management and defect tracking was neglected, and that team members' responsibilities were not clear [6]. Students need a systematic guidance during such a capstone project to lower the risk of these problems and ensure that the students are not distracted from their learning objectives and can perform "reflective" practice during problem solving.

A capstone project (i.e., a practicum project), conducted at the research group Software Engineering of the University of Kaiserslautern (UKL), Germany, is a project that lasts between two to three months and covers all the software development phases. The students get in contact with a real industrial customer. They develop either a complete smaller software system or a new component for an existing system by applying state of the art SE approaches. The goal of these projects is not to teach the students the usage of specific and complex SE tools, but to focus more on SE principles, methods, and techniques. Therefore, more and more easy-to-use, free- and open-source software (FOSS) have been used in the projects during recent years to keep the training period for getting familiar with the tools

Manuscript received December 15, 2006; revised June 20, 2007.

E. Ras is with the Department of Education and Training, Fraunhofer Institute for Experimental Software Engineering, 67663 Kaiserslautern, Germany (e-mail: eric.ras@iese.fraunhofer.de).

R. Carbon is with the Department of Product Line Architectures, Fraunhofer Institute for Experimental Software Engineering, 67663 Kaiserslautern, Germany (e-mail: ralf.carbon@iese.fraunhofer.de).

B. Decker and J. Rech are with the Department of Experience Management, Fraunhofer Institute for Experimental Software Engineering, 67663 Kaiserslautern, Germany (e-mail: bjoern.decker@iese.fraunhofer.de; joerg.rech@iese.fraunhofer.de).

Digital Object Identifier 10.1109/TE.2007.904580

as short as possible. In addition, academic institutions have to bear with shrinking budgets, and they are not able to acquire the newest commercial tools or the latest updates [7].

In this paper, a Wiki-based system is used to support the student to practice in a more reflective way and to train his or her metacognitive skills. The approach addresses the focus of this issue by presenting a concrete success story about using a Wiki in SE education for reflective practice and by providing first evaluation results from a case study. The results provide a foundation for more focused controlled evaluations in the future. Another intention for using a Wiki in the capstone project was to further investigate the impact of FOSS Wikis on the effectiveness and efficiency of specific SE activities.

Section II describes the potential of Wikis for higher education and SE activities. After a description of the capstone project setting, the conceptual infrastructure of the FOSS experience management system is described, and the process of how observations evolve to software experiences through reflection and abstraction is shown. Results from a recent case study show the utility of the Wiki for different SE tasks and provide evidence that a Wiki supports reflection activities through experience management.

II. Free- and Open-Source Wikis in Education

A few years ago, higher education had begun to explore the potential educational value of FOSS. Many educational institutions have been running installations of FOSS learning management systems, such as Sakai or Moodle, for several years. In addition, many academic institutions use FOSS for practicum projects, or they decide to let students participate in FOSS projects to allow them to participate in big projects [7]. Using FOSS is not only a matter for easily accessing the code and adapting the code, FOSS also affects the developed products and how the products are developed, still another reason why FOSS should be used in SE education [8].

Currently, the amount of social FOSS used in education is increasing constantly. FOSS supports people in connecting or collaborating through computer-mediated communication and in forming online communities [9]. Chat rooms and instant messaging are just two common examples of social software. Other social software such as Wikis, which have their origin in SE, have been recognized as beneficial knowledge management and group communication tools in the corporate world. A Wiki system, by definition, is "the simplest online database that could possibly work" [9]. Higher education is starting to investigate the potential of Wikis (and also blogs) regarding their support for learning, communication, and interaction processes [10], [11]. Wikis have been used as platforms for documentation, minutes, glossaries, or repositories for additional learning materials. Their advantages for higher education are fast installation, easy adaptation to educational purposes, no acquisition costs, and intuitive usage.

In addition, Wikis are also used in SE, and some of them have even been used in capstone projects: Trac is a Wiki written in Python that integrates an issue tracker and allows relating Wiki pages to issues, and vice versa [12]; Master of Arts in Special Education (MASE) (agile software engineering) offers plug-ins for agile software development, in particular for iteration planning and integration of automated measurement results [13]; SnipSnap, a Java-based Wiki, allows read-only integration of code documentation and offers support for the integration of Wiki entries into the integrated development environment Eclipse [14]; EclipseWiki is a Wiki integrated into the integrated development environment Eclipse [15]; Fitnesse is a Wiki-based test management framework that allows capturing and running test cases [16]. Recently, the *software organization platform* (SOP), an adapted Wiki for SE that has also been used for the case study in this article, has demonstrated its usefulness for stakeholder participation in requirements engineering [17].

III. Capstone Project

Since 2001, the research group Software Engineering at the University of Kaiserslautern has conducted an open-source (OS) capstone project once a year in cooperation with an industrial customer and Fraunhofer Institute for Experimental Software Engineering (IESE), Kaiserslautern, Germany.

A. Background of Students

The computer science (CS) undergraduate curriculum contains practical CS courses and technical and theoretical courses, mathematics courses, and an elective field of study such as electrical engineering or economics. When the students receive their bachelor's degree after three years, they are able to design small object-oriented systems based on the knowledge obtained in the courses Development of Software Systems I–III, and they are able to implement and test small software systems in teams of three to four people. If the students choose to focus on SE after the third semester of their bachelor studies, they have to enroll in the course Foundations of Software Engineering. In the two-year Master's program, the students can again choose for the SE option. These students can now sign up for the capstone project. They already know and understand about processes, methods, techniques, and tools that are used to develop large and complex software systems from their bachelor studies. At the end of their Master's studies, they get a Master's in CS.

B. Learning and Project Objectives

The main project goal is to fulfill the interests and requirements provided by the industrial customer and to get the students involved in a real industrial project. Students are fully responsible for eliciting the requirements, designing the prototype, and implementing and delivering the system on time. They will be faced with changing requirements, communication problems, etc. After the project, the students should be able to:

- know and understand the different roles and responsibilities in a software development project, especially the management-oriented roles such as the project manager;
- communicate and interact with a real customer;
- carry out project estimation (i.e., effort, time, quality);
- develop software in a team of 10 to 14 students;
- execute a well-defined software development process;
- judge existing OS components and understand the consequences when they are integrated into a product;
- understand the importance of software and experience documentation for future projects (i.e., to document observations and experiences during the project);

- be aware of their own thinking and decision-making processes;
- reflect about events and changes of situations that originate from performed actions.

C. Roles and Processes

A team of students works full time for two months in a laboratory environment between the summer and winter terms on the capstone project. Working between summer and winter terms was proven to be successful because they can fully concentrate on the project goals and tasks. Research staff of the working group Software Engineering at UKL and of Fraunhofer IESE coach the students during their work.

The students take over all roles in the project. The roles cover management-oriented roles like project manager, quality assurer, product manager, or experience manager and technical roles such as requirements engineer, architect, or tester. Each student must apply for one role during the kickoff meeting of the practical course. The capstone project follows an iterative development process with two iterations. Both iterations run through the phases requirements analysis, architecture and component design, implementation, and test.

D. Technical Environment of the Capstone Project

The technical laboratory environment, i.e., the tool environment, consists of FOSS and one commercial system:

- integrated development environment *Eclipse*;
- configuration management *Subversion* (product repository);
- experience management system *SOP*, based on *MediaWiki*;
- a commercial workflow management system.[1]

The usage of an integrated development environment (IDE) and a product repository is common sense. A workflow management system has been used to guide the students and assure that they follow a prescribed development process. The role of SOP will be described later.

IV. Supporting Reflection Through Experience Management

According to the last four learning objectives (see Section III-B), reflective skills are key skills to be learned by the students, in addition to the technical skills of SE. The same has been stated by Socha *et al.* for other engineering disciplines. They mention experiential learning as an effective way to teach these skills, with students continually going through a learning cycle: "practicing, reflecting on the difficulties, discovering new models (or having them introduced by facilitators or other students), and then practicing again" [18]. Reflection is a phase of the well-known learning cycle of Kolb and Fry [19], [20], who investigate the learning process related to learning from experiences and whose research has its foundation in the work of Lewin [21], Dewey [22], and Piaget [23]. Reflection is the prerequisite for learning from experience (e.g., in order to form abstract concepts) and for improving actions and professional practice [20]. Self-regulated learning theories focus on how students could activate, change, and maintain their learning practices. In recent years these theories have concentrated more on information processing and, in particular, on the metacognitive process of self-reflection [24]. Metacognition is related to higher order thinking that involves active control over the cognitive processes engaged in learning [25]. As Anderson and Krathwohl describe in their book on educational objectives, which revises the taxonomy of Bloom [26], metacognitive skills are skills that make the learners aware of their own knowledge and their ability to understand, control, and manipulate their own cognitive processes [27]. Hence, supporting self-reflective processes in a learning environment could enhance the learning benefit of the performed activities and give opportunity to review previous actions and decisions before proceeding to a next activity. Angelo and Cross provide an overview of how teachers can promote metacognition in a classroom [28].

The value of reflection has already been proven in situated cognition theory (e.g., *cognitive apprenticeship* and *anchored instruction*). The work of Schön highlights the importance of knowledge resulting from real experiences of professionals [29], [30]. Schön distinguishes between two types of reflection that facilitate the learning and activity of professionals: reflection-in-action and reflection-on-action. Short-term *reflection-in-action* is performed while people act and experience. The activity is reshaped while the activity is performed. *Reflection-on-action* is retrospective thinking about an experience after an activity or during an interruption. Other persons could be involved. The latter provides an understanding of practice and is a way practitioners may learn from their experience.

The accomplishment of teaching development and reflective skills is aggravated by the short, two month duration of the projects. This limited time requires the students to start as early as possible with the assigned tasks. How can one ensure that the customer requirements are met at the end of the project, that students practice previously mentioned skills, and that they learn efficiently based on their experiences made during the project?

A FOSS Wiki was adapted for experience in documentation, understanding, and sharing to teach reflective skills and to support coaching and guidance. In addition, the strategic goal was to build up an experience base and to gather data (effort, defects, etc.) systematically to support future OS capstone projects. Sections IV-A–C explain how experience management works in SE, how experience management can support reflection, and how observations and experiences are documented by the students.

A. Experience Management in Software Engineering

The reuse of existing knowledge and experience is one of the fundamental principles in many sciences. Engineers often use existing components and apply established processes to construct complex systems. Without the reuse of well proven components, methods, or tools, the systems have to be rebuilt over and over again.

During the last 30 years, the fields of software reuse and *experience management* (EM) have been gaining increasing importance. The roots of EM lie in experimental SE ["experience factory (EF)"], in artificial intelligence ("case-based reasoning"),

[1]The name of the commercial workflow management system is not mentioned because the choice is not motivated by the context of the capstone project.

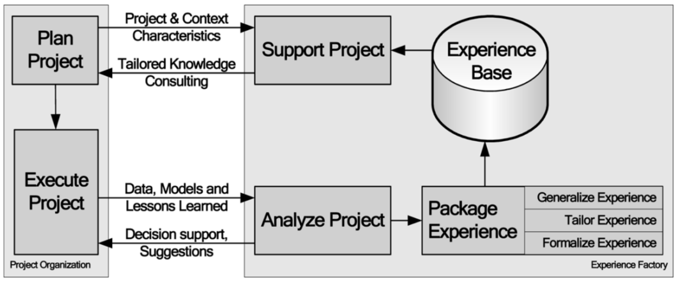

Fig. 1. Experience factory.

and in knowledge management. EM includes methods, techniques, and tools for identifying, collecting, documenting, packaging, storing, generalizing, reusing, adapting, and evaluating experience, and for development, improvement, and execution of all knowledge-related processes. The EF is an infrastructure designed to support experience management (i.e., the reuse of products, processes, and experiences from projects) in software organizations [31]. EF supports the collection, preprocessing, and dissemination of experiences. EF separates the project and the experience organization physically or at least logically as shown in Fig. 1. This separation is meant to relieve the project teams from the burden to find, adapt, and reuse knowledge from previous projects and to support them in collecting, analyzing, and packaging valuable new experiences that might be reused in later projects.

For example, if software engineers begin a project ("plan project"), they can use the experience factory to search for reusable experience in the form of reference architectures, design patterns, or process models based upon the project context. In the execution phase ("execute project"), the EF is used to retrieve experience "on demand" (e.g., to support decisions or reuse source code). Furthermore, during the project and at the end of the project, the project is analyzed (e.g., using a retrospective workshop) to extract reusable observations and experiences that might be useful in other projects.

B. Reflection Activities in the Project

Reflection activities are supported by the SOP, an adapted Wiki for information and experience management in software projects (Fig. 2). SOP provides information for guiding the students and documented observations and experiences, in particular during project execution. SOP serves as a means to capture observations and share all kinds of information relevant to the project [32]. Examples of the content of SOP are descriptions of roles and their responsibilities, process descriptions, document templates, documentation guidelines, observations and experiences on software engineering (SE) technologies, etc.

Most reflective activities refer to reflection-on-action, i.e., reflection after an activity or when an activity is interrupted (see dashed ellipses in Fig. 2). They help to reflect about recent observations/experiences, decisions made, and remaining problems. *Everyday stand-up meetings*, moderated by the coach, are conducted every morning (status of project, discussion about problems of the previous day, etc); an official *feedback meeting* with the customer takes place after the requirements are documented in the Wiki; an internal *review meeting* where all the project members attend occurs to discuss a first design of the architecture; after the test, the system is presented to the customer. The students receive valuable feedback about the system developed (focus on functionality); a *goal-oriented retrospective workshop* summarizes the first iteration regarding development process and technologies used, teamwork, roles, technical development infrastructure, and decisions about possible improvements for the second iteration. During or after these reflective activities, the students are asked by the *experience engineer* to document the observations and experiences into the Wiki. The experience engineer defines processes and guidelines on how to gather observations and experience, analyzes the findings, and packages them (see Section V for more details). Appropriate templates, offered by SOP, stimulate the students to reflect about the discussions and their own observations, and to self-reflect about their own knowledge, their learning process, and the applied problem solving strategies, etc. These templates use so-called *reflective questions* and *reflective prompts* that help the students to revise the details of the learning experience, to move toward critical thinking, and to create an action plan [33] after documenting the experience. Questions are of a more general nature, and prompts are more focused questions. Both types are related to the attributes of the templates (see Section V for details about the templates). The same reflective actions take place during the second iteration. At the end of the project, a final retrospec*tive workshop* is conducted. The purpose of both retrospectives is to gather more observations and experiences from students and to vote on lessons learned and reported.

SOP stimulates reflection-in-action by offering definitions, examples, detailed descriptions, etc. about processes, technologies, roles, etc. (more about the information structures in [32]). In addition to this information, SOP offers the reuse of already available observations, experiences, patterns, and laws that have been gathered in previous projects or during the first iteration. This information not only guides the students through their tasks but also leads them to reflections about current activities and to thoughts about how to adapt the activity.

C. Gathering Experiences

The observations and raw experiences from the projects are further refined by the students in the experience factory (Fig. 3). In SOP, customized templates can be designed by means of using an extended Wiki functionality that used the normal Wiki syntax cascading style sheets (CSS). Currently, SOP offers two templates for experience management: *observations* (O: name, situation, problem, solution) are suitable for easy and fast documentation of experiences; experiences (E_{SF}: name, situation, cause, solution, known exceptions, benefits, consequences, metadata) enable the student to provide more details about the experience and its context. In general, two main processes are used for refinement: 1) the *formalization* of the subjective and informal elements; and 2) the *generalization* of experiences to more abstract and generally applicable representations. Both require the student to perform reflection. As shown in Fig. 3, formalization is used to transform observations (O) into semiformal experiences (E_{SF}) and finally into formal experiences (E_{F}). Here, the aggregate or semiformal state refers to, for example, a structured template such as a pattern or experience

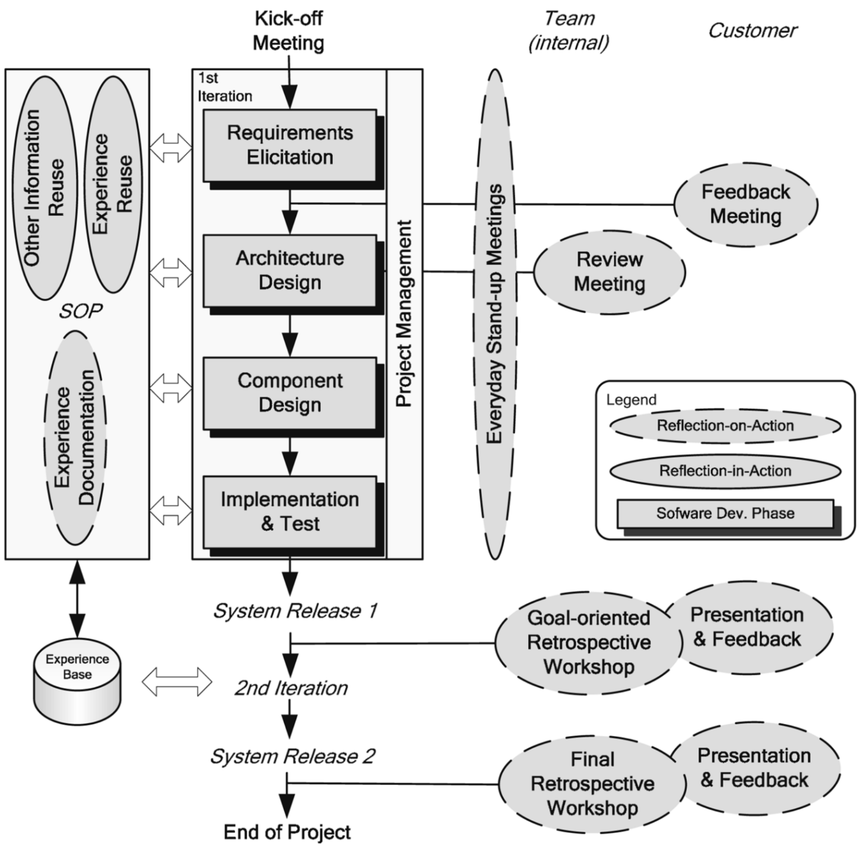

Fig. 2. Reflection during a capstone project.

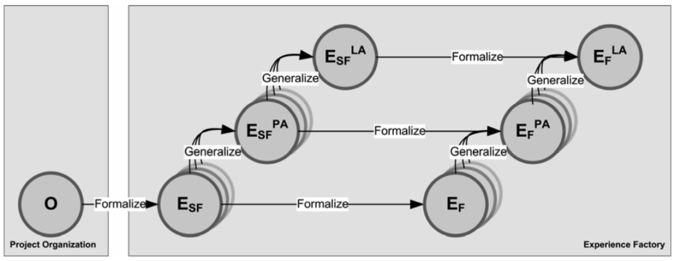

Fig. 3. Experience aggregate states.

template [34]. A formal experience represents a precise and clear definition, for example, based on predicate logic [35]. Generalization is used to summarize multiple project-specific experiences (E) into a pattern-aggregate of an experience (E^{PA}) and finally into a law-aggregate of an experience (E^{LA}). The core goal of this step is the decontextualization of the experience from its project, domain, (programming) language, or technology context. The aggregate state "pattern" represents, for example, a design pattern such as "abstract factory" [36], which is applicable not only in one project but in almost all object-oriented software systems. A law is a generally applicable statement, principle, or heuristic that is valid for all software systems, e.g., Brooks' Law "Adding manpower to a late project makes it later" [37]. The focus during the project was on observations and semiformal experiences. Pattern and laws are defined together with the coaching staff of the project.

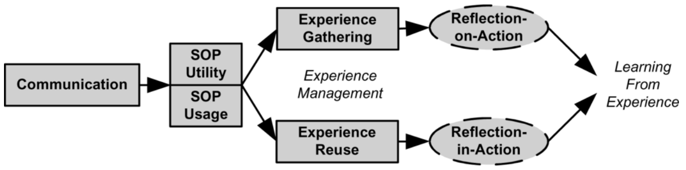

Fig. 4. Evaluation impact model.

Additional metadata for describing the experiences include: *type* (the type of an experience denotes its origin; typical values for type classification are: process, product, customer, organization, people, or project); *source* (the source of the experience: external, internal); *aggregate state* (denotes whether the element can be classified as an observation, experience, pattern or antipattern, or law); *formality* (formality describes the degree of structure, completeness, precision, and unambiguity using the levels: informal, semiformal, formal).

V. Evaluation

The goal of the evaluation was to assess the general utility of the FOSS platform for SE and experience management purposes and to investigate whether students learn from their experiences. The evaluation impact model in Fig. 4 shows which aspects were covered by the evaluation. Communication is an essential aspect in any short capstone project where the participants do not know each other beforehand. Technically supported communication has an impact on the usage and utility of the platform itself. In addition, SOP is intended to offer an experience management infrastructure. Experience gathering (i.e., through discussions, documentation, prioritizing, abstraction, and formalization) requires that the students reflect upon their experiences. Reflection-in-action is related to reusing existing experience of SOP in the current context. By reusing experience descriptions, the students reflect about their current activity and possible changes to it. Sufficient reflection ensures that the students learn from their experiences.

Each of the participants was asked to complete an online questionnaire covering the aspects of the impact model. Two types of questions have been used: most of the questions had to be answered by the degree of agreement on a four-point scale (i.e., fully agree, partially agree, partially disagree, and fully disagree); the second type of question was multiple choice, where zero or more options from a list could be selected. In addition, the contributions of the participants were analyzed using the *Wikistats* evaluation tool [38].

Thirteen of the 14 participants (93%) completed the questionnaire. Seventy-seven percent possessed programming experience, while 31% of all participants gained their experience in industrial organizations. All participants stated that they used the Wiki, and 69% used the Wiki regularly, which was also confirmed by the data of Wikistats.

Communication was rated by the participants as follows: The *improvement of information exchange* was rated within the team and among different teams. Except for the usage of contributions originating outside the team, all aspects achieved 100% full/partial agreement. The results imply that SOP improves the exchange of information, in particular within teams. Furthermore, the participants stated that by using SOP, they received *more information than* they would get *using only verbal communication*, and 23% of them agreed fully. The number of edits per day obtained via Wikistats confirmed that SOP improves communication. The number of edits per day reached a maximum of 175 during the requirements phase and dropped to about 100 edits per day later. (Because of privacy issues, a more detailed investigation per role or per person was not possible.)

To evaluate the *usage* and *utility* of SOP, the *Technology Acceptance Model* (TAM)[39] has been used, a thoroughly tested model to assess technology acceptance and of particular importance in capstone projects. These projects are expected to have a rather short run time. Furthermore, the students are expected to focus on the complexity of the project and should not be distracted by tool complexity. The *ease of use* was rated at a minimum of 77% (sum of answers fully/partially agree). An exception was the last item, (*sometimes, SOP behaves unexpectedly*), where 69% disagreed fully or partially. The top three items concerning *ease of use* were learnability, mastering of SOP features, and easy interaction. Based on this data, the conclusion was derived that SOP is easy to handle and to learn. *Perceived usage* was agreed with fully or partially by a minimum of 69%. The top item was access to information about the project, followed by general usage and contribution of project information. These results show that SOP can be used as a platform to exchange (capstone) project information. Concerning *self-predicted future usage*, more than 84% agreed fully or partially that they would use SOP for managing experiences in particular and for projects in general. Furthermore, 83% answered that they would use SOP even more if the support for document templates and visualization of related articles were improved. The answers in the questionnaire are backed up by the data gained via Wikistats: More than 360 articles were created during the capstone project, most of them during the requirements phase between mid-August and the beginning of September. Seven participants were also rated as very active (i.e., 100 or more edits within one month).

Perceived utility as the second part of the TAM provided a utility evaluation of SOP independent of concrete SE tasks. Six of nine items had a rating of more than 50% full or partial agreement. The minimum perceived utility was performance with 38% partial agreement. The top three were general utility, quality of products, and improvement in productivity. SOP is, in general, perceived as useful (und usable) in capstone projects. This rating might be caused by important features missing in SOP during the run time of the capstone project. An example was the missing export function from Wiki-Pages to office documents, which was needed several times in the capstone project for reporting to the customer.

The utility of SOP was also investigated regarding its support for SE tasks. The *general utility for SE tasks* was rated using multiple choice. Five of eight tasks were rated useful by over 50% of the participants (i.e., experience management, requirements, design, quality assurance, and project management). Therefore, the general utility of SOP is high. A reason for the lower rating of the other tasks might be that they are not supported by dedicated functions within the Wiki.

The perceived *effort savings* were captured according to the tasks in a software project (requirements, design, quality assurance, integration and test, implementation, product and configuration management, project management, and experience management). Concerning these tasks, more than 50% of the participants agreed fully or partially on four tasks that by using SOP they would save effort (Fig. 5). The top three tasks were experience management, project management, and configuration management. For experience management, there was no disagreement that SOP is actually saving effort. This agreement shows that SOP needs to support hands-on, software development tasks better, and that SOP provides support in managerial tasks. Again, the low rating of the effort saving effects might be caused by SOP not supporting these tasks—with one exception: As mentioned above, the effort during the requirement phase was increased because of creation of office documents by hand. In the current SOP version, this export can be performed automatically.

Because of the difficulty to assess the amount of *reflection-in-action*, other indirect measures regarding the quantity and quality of experience reuse were used. The perceived *reusability of experience* was rated as follows: That the captured experience will be reusable within the team was agreed with fully or partially by 85%. Seventy-five percent stated that the experience would also be reusable by other teams. That the experience is of no use was disagreed fully and partially by 64%. This report shows that the experience reuse by students is accepted and supposed to be helpful even across capstone

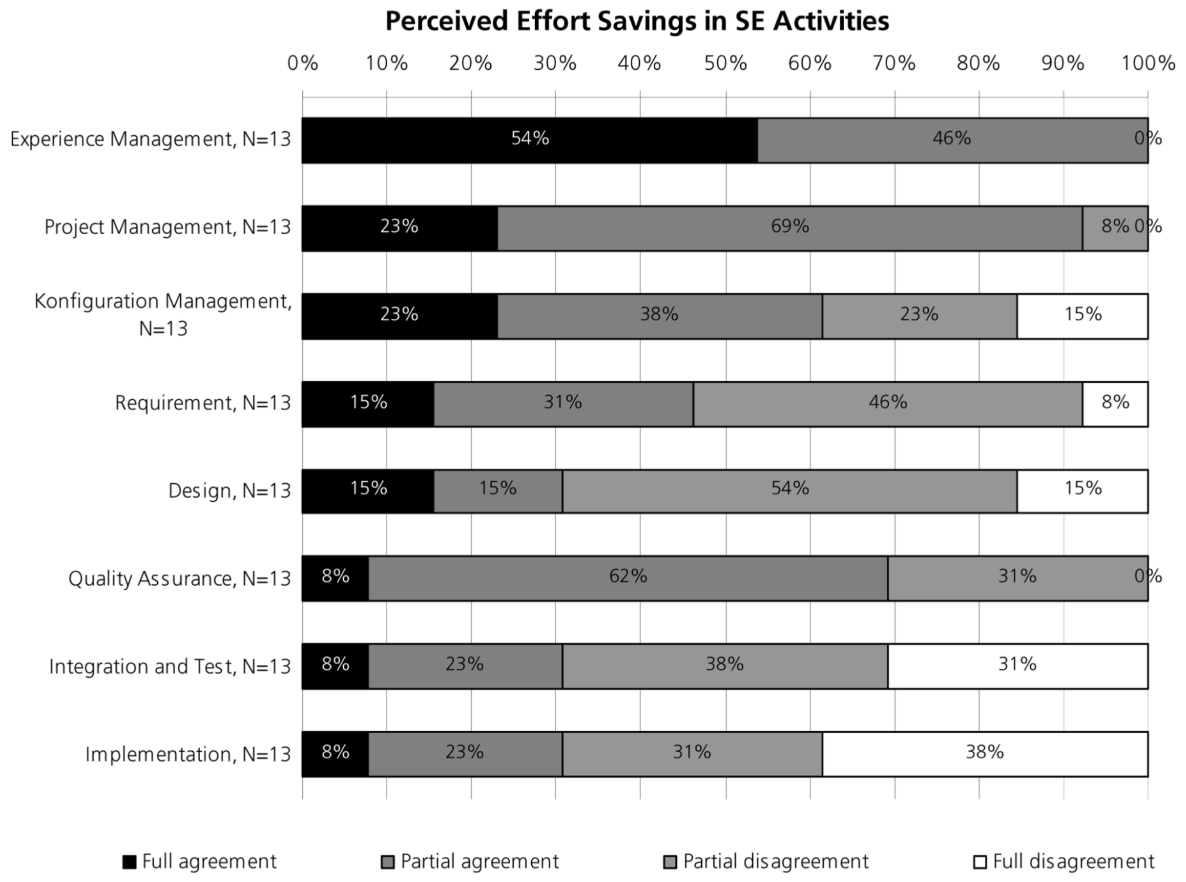

Fig. 5. Perceived effort savings in SE activities.

projects. The *quality of reused experience* was rated using an inverse scale, since the item asked for quality deficits. All quality issues (understandability, too abstract, not applicable, missing context description, too specific) were disagreed with partially and fully by at least 61%. The largest deficits identified were in the areas of context and abstraction level. The students had problems adapting the documented experience to their own situation. However, the results implied that the overall quality of experience was sufficient, but improvement is necessary. The *potential utility of types of experience*, i.e., which type of experiences would have been helpful in the project, provided the following results. All types had at least 53% full or partial agreement, and no full disagreement. The top three types were experiences with tools (100%), processes (85%), and SE methods (92%). This feedback will be used to prioritize the types of experience which should be offered to the students in future capstone projects.

Reflection-on-action is supported by the different types of meetings and technically by SOP (i.e., experience gathering). The following questions refer only to SOP. Each documentation of an observation or experience requires self-reflection of the students. Therefore, the number of documented items can be used as an indirect measure for reflection. During the project 178 observations and five experiences were gathered. Wikistats showed the following results: the articles were used throughout the capstone project, and the document structure remained stable. That *experience management is supported in general* by SOP was agreed with partially/fully by 92%. The top three types of experience management are lessons learned about tools (62%), processes (46%), and products/project management (38%). Other types of experience such as SE techniques, SE methods, SE principles, and fellow project team members achieved 31%.

Concerning *newly acquired topics* (Fig. 6), the top three items were products, fellow project workers (employees), and tools concerning full or partial agreement. Five of eight topics were rated better than 50%. New topics about processes were used by 53%, while new project management topics were used by 46%. This data indicates that many new topics were used in general. The topics of particular relevance in capstone projects (processes, project management) also had a fair rating of about 50%. However, further improvement should be made.

VI. Conclusion and Future Work

Wikis have several advantages for higher education, such as fast installation, easy adaptation to educational purposes, no acquisition costs, and intuitive usage. Many of these facts have been proven by the application of SOP. The evaluation showed that SOP is a suitable technical environment for gathering and reusing experiences, supporting requirements engineering in particular [17], accelerating feedback cycles, and for improving communication among students. The results of this case study help to convince other departments of the UKL to use FOSS for educational purposes. Many observations and experiences

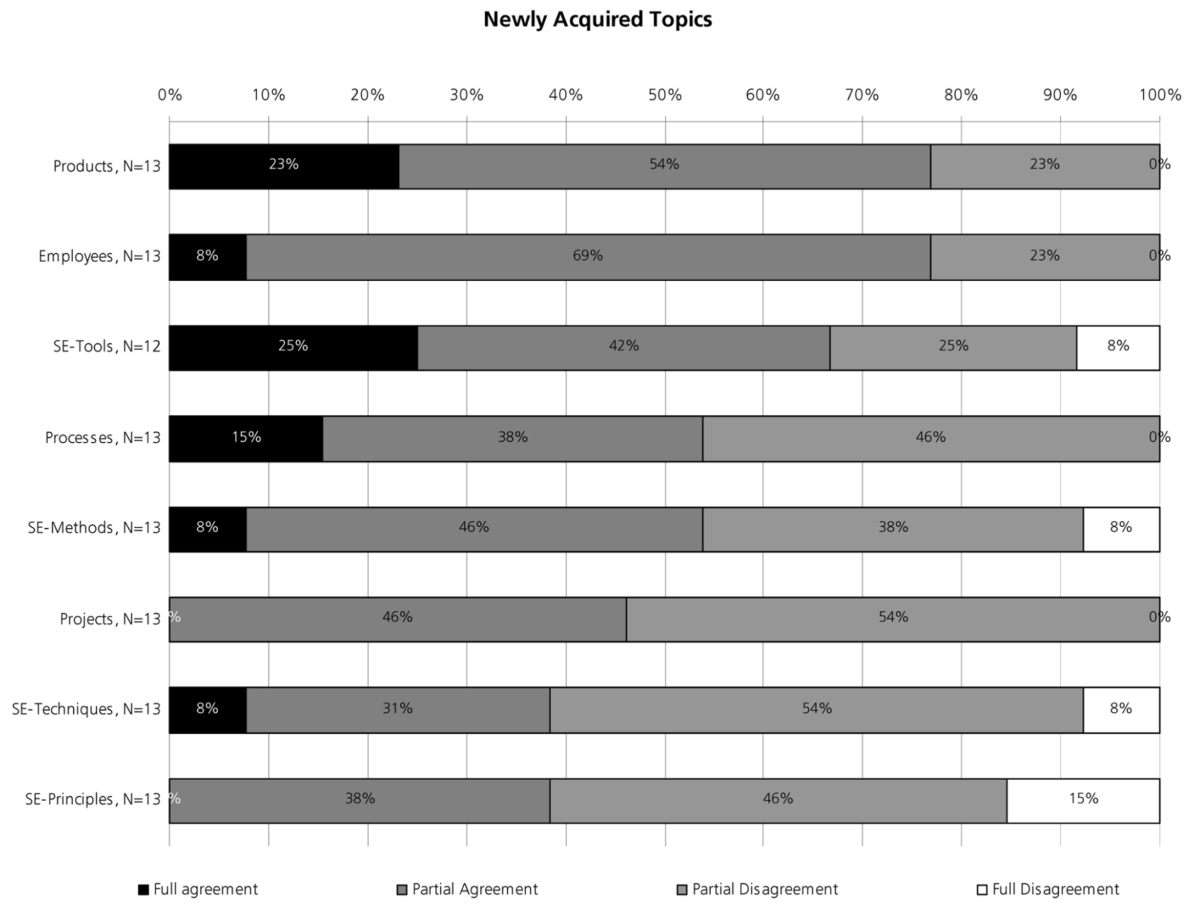

Fig. 6. Newly acquired topics.

have been gathered for future projects. Thus, students developed their reflective skills and learned from their experiences. The evaluation showed that standard Wiki functions support communication and information sharing, but a Wiki must be further adapted to the educational objectives and tasks of a capstone project. Without these extensions, no significant effort savings will be obtained in specific SE tasks. Wikis are not a replacement for continuous coaching and guidance of the students by educational staff; they are merely a means to support these activities.

One limitation of the evaluation is that no direct measures related to learning or reflection in particular have been used. In addition, the evaluation did not contain a formal assessment to find out whether the learning objectives have been met. Therefore, two controlled experiments will be conducted in 2007 that especially focus on identifying factors that have an impact on learning outcome (i.e., response variable) by using SOP. A first fractional factorial design will serve to find the two factors that have the biggest impact on the response variables (i.e., higher number of groups with few subjects in each group) and to build a baseline for the hypotheses of the second experiment. A second factorial experiment will use the identified factors (with a maximum of two alternatives each) to identify significantly the impact on the response variable (i.e., small number of group with higher number of students). The results of these experiments will be published in 2008.

Currently, SOP is further extended to serve also as a learning platform to integrate knowledge management with e-learning [40]. SOP will be able to provide learning content to the students during work. In addition, special emphasis will be put on the improvement of experience reuse by means of learning spaces [41].

To support further research and development of FOSS systems such as SOP, the authors decided to provide a stable version of SOP to the public in summer 2007. Other research institutions could then further develop and evaluate SOP for their own educational purposes and interests.

REFERENCES

[1] M. Shaw, J. Herbsleb, and I. Ozkaya, “Deciding what to design: Closing a gap in software engineering education,” in *Software Engineering Education in the Modern Age*. Berlin, Germany: Springer-Verlag, 2006, vol. 4309, pp. 28–58.
[2] D. L. Evans, B. W. McNeill, and G. C. Beakley, “Design in engineering education: Past views of future directions,” *Eng. Educ.*, vol. 80, no. 5, pp. 517–22, 1990.
[3] Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, Joint Task Force on Computing, Curricula, 2004.
[4] H. van Vliet, “Reflections on software engineering education,” *IEEE Softw.*, vol. 23, no. 3, pp. 55–61, May-Jun. 2006.
[5] L. J. Burnell, J. W. Priest, and J. B. Durrett, “Teaching distributed multidisciplinary software development,” *IEEE Softw.*, vol. 19, no. 5, pp. 86–93, Sep.–Oct. 2002.
[6] D. A. Umphress, T. D. Hendrix, and J. H. Cross, “Software process in the classroom: The capstone project experience,” *IEEE Softw.*, vol. 19, no. 5, pp. 78–81, Sep.-Oct. 2002.

[7] K. Toth, "Experiences with open source software engineering tools," *IEEE Softw.*, vol. 23, no. 6, pp. 44–52, Nov.-Dec. 2006.
[8] D. Spinellis and C. Szyperski, "How is open source affecting software development?," *IEEE Softw.*, vol. 21, no. 1, pp. 28–33, Jan.-Feb. 2004.
[9] B. Leuf and W. Cunningham, *The Wiki Way. Collaboration and Sharing on the Internet*. Reading, MA: Addison-Wesley, 2001.
[10] J. Williams and J. Jacobs, "Exploring the use of blogs as learning spaces in the higher education sector," *Aust. J. Educ. Technol.*, 2004.
[11] S. Reinhold, "WikiTrails: Augmenting Wiki structure for collaborative, interdisciplinary learning," in *Proc. Int. Symp. Wikis*, Odense, Denmark, Aug. 21–23, 2006, pp. 47–58.
[12] Integrated SCM and Project Management, Trac, 2007 [Online]. Available: http://www.edgewall.com/trac
[13] Agile Software Engineering, MASE, 2007 [Online]. Available: http://sourceforge.net/projects/mase
[14] The Easy Weblog and Wiki Software, SnipSnap, 2007 [Online]. Available: http://www.snipsnap.org
[15] An Open Development Platform, Eclipse, 2007 [Online]. Available: http://www.eclipse.org
[16] The Fully Integrated Standalone Wiki and Acceptance Testing Framework, Fitnesse, 2007 [Online]. Available: http://www.fitnesse.org
[17] B. Decker, E. Ras, J. Rech, P. Jaubert, and M. Rieth, "Wiki-based stakeholder participation in requirements engineering," *IEEE Softw.*, vol. 24, no. 2, pp. 28–35, Mar.-Apr. 2007.
[18] D. Socha, V. Razmov, and E. Davis, "Teaching reflective skills in an engineering course," in *Proc. Amer. Soc. for Engineering Education Annu. Conf. Exposition*, Nashville, TN, Jun. 2003, pp. 10317–10336.
[19] D. A. Kolb and R. Fry, "Toward an applied theory of experiential learning," in *Theories of Group Process*, C. Cooper, Ed. New York: Wiley, 1975.
[20] D. A. Kolb, *Experiential Learning: Experience as the Source of Learning and Development*. Englewood Cliffs, N.J.: Prentice-Hall, 1984.
[21] K. Lewin, *Field Theory in Social Science*. New York: Harpers & Row, 1951.
[22] J. Dewey, *Experience and Education*. New York: Collier, 1938.
[23] J. Piaget, *Psychology and Epistemology*. Middlesex, U.K.: Penguin, 1971.
[24] B. J. Zimmerman and D. H. Schunk, *Self-Regulated Learning and Academic Achievement: Theoretical Perspectives*, 2nd ed. Mahwah, NJ: Erlbaum, 2001.
[25] G. Schraw and D. Moshman, "Metacognitive theories," *Educ. Psychol. Rev.*, vol. 7, pp. 351–371, 1995.
[26] B. S. e. Bloom, M. D. Engelhart, E. J. Furst, W. H. Hill, and D. R. Krathwohl, *Taxonomy of Educational Objectives; the Classification of Educational Goals*, 1st ed. White Plains, NY: Longman, 1956.
[27] L. W. Anderson and D. R. Krathwohl, *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. White Plains, NY: Longman, 2001.
[28] T. A. Angelo and K. P. Cross, *Classroom Assessment Techniques: A Handbook for College Teachers*, 2nd ed. San Francisco, CA: Jossey-Bass, 1993.
[29] D. A. Schön, *The Reflective practitioner: How Professionals Think in Action*. London, U.K.: Arena, 1995.
[30] D. A. Schön, *Educating the Reflective Practitioner: Toward a New Design for Teaching and Learning in the Professions*, 1st ed. San Francisco, CA: Jossey-Bass, 1990.
[31] V. R. Basili, G. Caldiera, and H. D. Rombach, "Experience factory," *Encycl. Softw. Eng.*, vol. 1, pp. 469–476, 1994.
[32] J. Rech, E. Ras, and B. Decker, "Riki: A system for knowledge transfer and reuse in software engineering projects: Strategies beyond tools," in *Open Source for Knowledge and Learning Management*, M. Lytras and A. Naeve, Eds. Hershey, PA: Idea, 2006.
[33] M. Scardamalia and C. Bereiter, "Fostering the development of self-regulation in children's knowledge processing," in *Thinking and Learning Skills: Research and Open Questions*, S. F. Chipman, J. W. Segal, and R. Glaser, Eds. Mahwah, NJ: Erlbaum, 1985, vol. 2, pp. 563–577.
[34] A. Kamel, M. Chandra, and P. Sorenson, "Building an experience-base for product-line software development process," in *Proc. 4th Int. Conf. Case-Based Reasoning*, Vancouver, BC, Canada, 2001, pp. 13–20.
[35] T. Taibi, *Design Pattern Formalization Techniques*. Hershey, PA: Idea, 2007.
[36] E. Gamma, J. Vlissides, R. Johnson, and R. Helm, *Design Patterns: Elements of Reusable Object Orientated Software*. Reading, MA: Addison Wesley Longman, 1998.
[37] A. Endres and H. D. Rombach, *A Handbook of Software and Systems Engineering: Empirical Observations, Laws, and Theories*. Harlow, U.K.: Pearson, 2003.
[38] Analyzing and Visualizing the Semantic Coverage of Wikipedia and Its Authors, Wikistats, 2007 [Online]. Available: http://arxiv.org/abs/cs.IR/0512085
[39] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *Mange. Inf. Syst. Q.*, vol. 13, pp. 319–340, 1989.
[40] J. Rech, E. Ras, and B. Decker, "Riki: A system for knowledge transfer and reuse in software engineering projects," in *Open Source for Knowledge and Learning Management*, M. Lytras and A. Naeve, Eds. Hershey, PA: Idea, 2007, pp. 52–122.
[41] E. Ras, J. Rech, and B. Decker, "Workplace learning in software engineering reuse," in *Proc. Int. Conf. Knowledge Management, Special Track: Integrating Working and Learning*, Graz, Austria, 2006, pp. 437–445.

Eric Ras received the B.S. (Vordiplom) and M.S. (Diplom) degrees in computer science from the University of Kaiserslautern, Germany.

He is currently Scientific Coordinator of the International Distance Learning Program, Software Engineering for Engineers, at the University of Kaiserslautern. He organizes a workshop on learning-oriented knowledge management and KM-oriented e-learning. He is a program committee member of different workshops and conferences in the domain of software engineering, e-learning, and knowledge management. He has worked as a Scientist on different public and industrial projects in the domain of knowledge management, e-learning, and document engineering at the Fraunhofer Institute for Experimental Software Engineering. He has gathered experience in reuse-based learning material production and work-process oriented vocational training methods. He has worked intensively with current e-learning standards and tools and has done research in the domain of social free- and open-source software.

Ralf Carbon received the B.S. (Vordiplom) and M.S. (Diplom) degrees in computer science with a minor in economics from the University of Kaiserslautern, Germany.

He is currently a Researcher at Fraunhofer Institute for Experimental Software Engineering (IESE), Kaiserslautern, Germany. He is in charge of an empirical software engineering laboratory and is assigned to research and industrial projects in the competence center, "Virtual Office of the Future." His research focuses on product line engineering, agility and flexibility, service-oriented computing, and open-source software. He has been involved in open-source evaluation projects with the University of Kaiserslautern since 2002. Before joining Fraunhofer IESE in 2005, he worked in the Software Engineering Research Group, University of Kaiserslautern.

Björn Decker received the B.S. (Vordiplom) and M.S. (Diplom) degrees in computer science with a minor in economics from the University of Kaiserslautern, Germany.

He is currently a Project Manager and Scientist at the Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany. His research mainly concerns experience management, the usage of Wikis in software engineering, business process management, and collaborative maintenance. He has authored a number of papers on software engineering and knowledge management. He is organizing the 2007 German Workshop on Experience Management and is a program committee member of different workshops and conferences.

Jörg Rech received the B.S. (Vordiplom) and M.S. (Diplom) degrees in computer science with a minor in electrical science from the University of Kaiserslautern, Germany.

He was a Research Assistant to Prof. Dieter Rombach in the Software Engineering Research Group, University of Kaiserslautern. He is currently a Project Manager and Scientist at the Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany. His research mainly concerns defect discovery, (anti-) patterns, model driven software development, knowledge discovery in software repositories, code mining, code retrieval, software analysis, software visualization, software quality assurance, and knowledge management. He has authored a number of papers on software engineering and knowledge management.

Mr. Rech is a member of the German Computer Society [Gesellschaft für Informatik (GI)]. He is also the speaker of the GI working group on architectural and design patterns.