Reputation-based Self-management of Software Process Artifact Quality in Consortium Research Projects

Christian R. Prause
Fraunhofer FIT
Schloss Birlinghoven, Sankt Augustin, Germany
christian.prause@fit.fraunhofer.de

ABSTRACT

This paper proposes a PhD research that deals with improving internal documentation in software projects. Software developers often do not like to create documentation because it has few value to the individual himself. Yet the purpose of internal documentation is to help others understand the software and the problem it addresses. Documentation increases development speed, reduces software maintenance costs, helps to keep development on track, and mitigates the negative effects of distance in distributed settings. This research aims to increase the individuals' motivation to write documentation by means of reputation. The CollabReview prototype is a web-based reputation system that analyzes artifacts of internal documentation to derive personal reputation scores. Developers making many good contributions will achieve higher reputation scores. These scores can then be employed to softly influence developer behavior, e.g. by incentivizing them to contribute to documentation with social rewards. No strict rule enforcement is necessary.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Documentation; D.2.9 [**Management**]: Software quality assurance

General Terms

Documentation, Human Factors, Management, Measurement

Keywords

software quality, documentation, collaboration, reputation

1. INTRODUCTION AND BACKGROUND

Source code is the most precise description of a software [9]. It contains all the detailed knowledge. Other artifacts of the software process like requirements, designs, rationale and so forth are no less important to humans for understanding. This *internal documentation* encompasses all information that originated from the development of a software product. It is bound for developers and maintainers, and is internal to the organization and project [11]. (External documentation is not treated here.) Due to complexity, a large software system is difficult to understand. But a certain understanding is necessary to modify it. Working on it is therefore a continuous learning process. Documentation is the learning material that provides developers with the knowledge that enables them to modify and extend a software [18]. It serves as a reference to the original developer but is especially valuable to others [25].

In general, research must deliver results that are well usable in the next phase of the innovation process. Fundamental research delivers proven concepts; applied research creates prototypes and runs pilots to demonstrate that a certain design is suitable and works as expected. Prototypes bridge the gap between research and industrial products.

Prototypes are cheap and light-weight versions of software that can later be extended into full-fledged products [13]. They may have bugs, miss features and look crude but the knowledge obtained through them must be preserved. The nearer development draws to its product phase, the more important becomes engineering and the precise knowledge of construction details and design rationale. Only then is incremental refinement possible, and can lead to products.

The majority of projects co-funded under the European Union (EU) Framework Programmes (FPs) are applied research projects, i.e. projects conducting ambitious research with the goal of subsequent industrial exploitation. Research consortia bring together universities, research institutes and small to large industry partners from several different countries. However, documentation is a particularly intricate matter in this environment (see Section 2).

The thesis research deals with improving software development in distributed, multi-partner research projects by motivating developers to contribute more to documentation. It applies reputation and economic theory elements to the software process to enable self-management of documentation. Documentation is considered as a common property that is subject to the "tragedy of commons" (see [8]) – the decay of a valuable resource due to a misalignment between individual and social cost: Documentation has few value to the individual developer but costs him his own precious time to produce [25]. Yet for the whole team, the total "social" cost of not having good documentation is much higher [24].

There are two strategies for preventing poor documentation quality: one is to enforce strict regulations on the usage of the common property, the other one is to *internalize* social

ESEC/FSE'11, September 5–9, 2011, Szeged, Hungary. ACM.
DOI: https://doi.org/10.1145/2025113.2025166
Self-archived at: https://www.drprause.de/publications/

cost [4]. Most current approaches follow the first strategy (see Section 3). This research focuses on the second strategy.

The idea is to assign personal responsibility for collaboratively created and used artifacts (source code, wiki pages, ...), and thus internalize common property. *Internalizing* means that the social cost of degeneration is partially transferred to individuals that consequently have a higher interest in preventing degeneration [4]. The CollabReview platform was developed as tool-support for automatically internalizing common properties in a software project (Section 4).

Improving internal documentation reduces project costs and increases quality, [24] and thereby makes industrial exploitation easier and more probable. Compared to approaches based on regulation, internalizing common properties could lessen the need for strict quality prescriptions and instead give developers more flexibility. Additionally, fewer quality control would be necessary, reducing quality control costs and, more importantly, being more suitable in consortium-based projects with flat hierarchies (see Section 6).

2. PROBLEM DESCRIPTION

A lack of high quality documentation is a pervasive problem in software projects. For example, a study of 500 data processing organizations finds that poor documentation is one of the biggest problems [14]. The role and maintenance of documentation is either poorly understood, or documentation does not exist at all because of other project pressures [7, 5]. Software developers are famous for their dislike of documentation, often trying to avoid doing it altogether [25]. Wray asks "Why do we persist in poor programming practices when we know they're poor?" [30]. Boehm calls it the "cowboy role model" of young developers [2]. Even for prototype projects in research, insufficient documentation is a problem when a later project continues the initial research but the original developers and their knowledge are not available anymore [21].

During the last years, I gained insight into a family of projects where improving software processes is difficult. With its FPs, the European Commission (EC) co-funds hundreds of so-called "cooperative projects" with an average cost of more than 5M Euros each year. These projects are signified by a high degree of distributedness because they involve multiple partners by definition. The partners come from several countries, and have different organizational backgrounds: small- and medium-sized enterprises, large corporations, universities and research institutes [22]. Obtaining high quality documentation is challenging but nonetheless important for the demanded industrial exploitation.

In a consortium project, partners will try to minimize their own problems at the expense of others [5] and prioritize practices differently. For example, writing documentation can be considered overly bureaucratic by some, too clumsy by others or just unsuitable. However, as consortium management has a weaker position here than project management in projects conducted by a single entity, agreed documentation standards are difficult to establish.

For example, towards the end of one project, telephone conferences and email discussions were held frequently to try to jointly reconstruct knowledge of the actual workings of the system. Meanwhile, since commercial exploitation opportunities increased, even a reimplementation was discussed. A software system from an earlier project was in such a bad condition that one developer, when leaving for a new job, recommended "to fight tooth and nail" to never have to maintain that system. In another project, some partners had never worked with a wiki before, and prevented that it was established as a primary point of reference. So it contained only few documentation while other documentation was distributed over reports delivered to the EC. One partner resignedly noted that getting documentation from partners was like "trying to get blood out of stone". Developers mostly avoided writing messages when committing source code to revision control, or rarely committed twice a year. Entering bug reports into a tracker was seen as bureaucratic. Therefore not a single report was filed, although issue tracking is pivotal for knowledge management [1].

Poor quality and the non-existence of certain software process artifacts means that knowledge is not (adequately) preserved. When it fades from the developers minds, it is either lost forever, or has to be recovered or recreated in a costly process. For instance, understanding a software architecture from source code only is extremely difficult, guessing non-functional requirements from an implementation comes close to impracticality, and throwing away code means to lose all the implementation subtleties. The thorough use documentation therefore actually speeds implementation and reduces total cost in the long run [24].

3. RELATED WORK

Organizational operation is characterized by the dimensions certainty and agreement. Where there is low certainty of the effects of actions and low agreement between agents about the nature of an issue, processes are difficult to establish, or can even be harmful [27]. EU projects operate in this area. Heavy-weight processes are therefore not suitable.

Approaches like the "Don't repeat yourself" [26] principle aim at reducing the developers' effort when creating documentation. Similarly, tools like Javadoc can generate certain documentation by analyzing source code, or relate existing pieces of documentation [16]. However, manually created documentation that explains the complex backgrounds of software will always be necessary [24].

It seems that a certain amount of continuous pressure to force a change in developer behavior — either through rules and enforcement, or incentives — is necessary [20]. Pair programming has been identified as a source of pair-pressure that reduces bad habits [29]. Also, Extreme Programming reduces the need for documentation through an active direct exchange of knowledge in the team, and with customers. But in EU projects, team sizes at each partner's site are often less than five people [22]. Here, knowledge transfer between sites has to take different forms. Similarly, coding and documentation rules are difficult to enforce across sites.

Relevant to this research are furthermore concepts that attempt to alter user behavior through rewards. Successful examples exist in very different domains: One example is the "Thief of the Week award" that rewards developers who re-use code instead of developing it anew [15]. In pervasive computing, Yamabe et al. showed that micro-payments can have a significant impact on user behavior [31]. Since the advent of the Internet, reputation systems have become an important factor in online communities to alter user behavior [12]. Social rewards are applied to wikis in work places to motivate employees to contribute [10, 6]. Although metrics are important in software development, they have to be used with care because they might be perceived as threats

to developers' careers. Also, metrics might fail and not tell the truth, or be manipulated by developers [28].

4. THESIS STATEMENT

There is no universal documentation method. This holds true for research projects, too, where the predominant form of organization is self-organization. Only the individual developer can effectively decide what is worth documenting and what not. As the purpose of documentation is to impart knowledge in its audience, it has to be judged based on this ability. It may, for instance, be incomplete and still be appreciated, as long as the necessary information is conveyed [7]. By adding a little motivation to do documentation, developers may be willing to contribute more. I claim that...

> *"Internalizing software process artifacts with a reputation-based software tool in combination with suitable rewards, encourages the individual developer himself to care more for the quality of the project team's common property."*

To allow researching this claim, the CollabReview platform has been developed. CollabReview is a reputation system that analyzes how developers interact with the common properties. A developer who often makes high-quality contributions will consequently receive a high reputation score. The scores are then exploited to influence developer behavior. For example, publishing all developers' scores to the team can reward well-behaving developers. But also social games, computing personal technical debt with respect to documentation [3], or tangible rewards are possible.

In order to compute reputation scores, CollabReview assesses the quality of each artifact of documentation, and determines who contributed to it. The quality of an artifact of documentation is determined by voting (called review). Feedback from developers as well as results of automatic analyses (e.g. static analysis) are collected, and then averaged to compute the quality rating for that artifact. To account for out-of-date reviews, reviews are weighted according to their timeliness. Timeliness is based on how much the reviewed artifact has changed since the review [19].

The developers' contribution to an artifact is calculated by CollabReview through authorship. Text that is contributed to the artifact by a developer, is said to be authored by that developer. A developer who owns much of the text of an artifact, has a high responsibility for the artifact. For instance, if half of the text of an artifact was written by a certain developer, then that developer will have half of the responsibility for the artifact. CollabReview compares different revisions of an artifact to determine authorship.

Each new revision of an article is compared with several of its probable ancestors. When an optimal candidate has been found, the algorithm determines an author for the newly added or modified lines; unmodified lines will retain their authorship information. If a non-trivial line (e.g. one that contains only whitespace) is found to be a clone of an already existing line, authorship information is copied from there. Otherwise the revision's creator is author of the line [17].

5. APPROACH AND CONTRIBUTIONS

The thesis research interconnects different scientific areas. At a technical level, there are analyses of project common property to obtain authorship, and quality information through static analysis and reviews. At a higher level, software processes, collaboration, motivation and economic aspects must be considered. A solution cannot be purely technical, theoretical or conceptional.

The research therefore loosely follows a user-centered design approach. Yet the proposed solution was not explicitly demanded by users. Instead, it is developed to exploit potential for improvement. However, concept and prototype are developed iteratively with frequent user involvement.

Years of project experience led to the observation of problems as described in Section 2. Conventional software process improvement techniques were difficult to establish in these environments. An empirical study involving 100 respondents from 50 EU projects was conducted to define a frame of reference for the personal project experiences [22].

An initial set of requirements was gathered from personal experience. They were used to draft a solution concept, and were discussed with colleagues. Refinements to the concept were based on existing literature [20]. After this, the initial CollabReview prototype was implemented [19]. The concept was presented and discussed in three expert rounds to generate conceptual feedback. This feedback led to refinements of the concept, and further requirements and constraints.

CollabReview was tested with source code from two real projects (one EU and one open source project). The goal was to show that the prototype can handle real-world data and produce sensible results. Adaptations were necessary to make the prototype capable of handling the large datasets that occur in reality. A source code evolution algorithm was developed to improve handling of moved code, and to increase the tampering resistance of CollabReview's authorship analysis [17]. For source code, CollabReview's way of computing reputation has been validated. Validation was possible because static analysis can cheaply generate objective quality data. The validation first computed developers' reputation scores, and then used the reputation scores to predict source code quality in a 10-fold cross validation. The results showed a significant, substantial correlation of $r \approx 0.4$. While prediction is not highly accurate, it shows that reputation scores work as indicators for prediction, and are therefore sound. (Publication pending.)

In parallel to this, a survey of rewards used in practice was conducted. From this survey, twelve dimensions of the *incentives space* were identified and aligned with the responses of 16 subjects working in EU projects [23]. This information directed the search for a suitable rewarding mechanism.

Moknowpedia, the knowledge management wiki of the Mobile Knowledge workgroup, was chosen as a first field-test environment for the CollabReview platform. The prototype was adapted to process wiki articles instead of source code. Its user interfaces were reworked for seamless integration with the wiki. In a two months trial, the effects of the extensions to Moknowpedia were evaluated. Significant improvements to article quantity and quality were measurable as well as perceived by the users [6]. The reception of the platform was mostly positive, and it keeps being in use even after the evaluation has ended. The experiences gathered with the enhanced Moknowpedia informed further design decisions. Work is in progress to improve the reputation-based rewarding scheme to make it more effective.

6. EVALUATION

This section details the plans for the final evaluation of the concept. (Section 5 lists validations and evaluations that

were already conducted.) The final evaluation is scheduled for 2011. It concludes the thesis research with field-tests in EU projects, and a laboratory experiment.

CollabReview is currently being integrated with the GForge software project infrastructure that is used in several EU projects. The integration with GForge will enable field-tests in two EU projects with up to 100 developers from different European countries. In these field-tests, the effects on both source code quality and wiki articles are investigated, and developer reactions are studied.

The laboratory experiment will test the researched concept under controlled conditions. Two groups of users, control groups and experimental groups, will implement the same, specified, small software. While the experimental group will be using CollabReview, CollabReview will not be installed in the control group.

Acknowledgment

This work is co-funded by the *ebbits* EU FP7 project under grant agreement no. 257852.

7. REFERENCES

[1] D. Bertram, A. Voida, S. Greenberg, and R. Walker. Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams. In *CSCW*, pages 291–300. ACM, 2010.

[2] B. Boehm. Get ready for agile methods, with care. *Computer*, 35:64–69, 2002.

[3] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, and N. Zazworka. Managing technical debt in software-reliant systems. In *FoSER*. ACM, 2010.

[4] R. H. Coase. The problem of social cost. *The Journal of Law and Economics*, 3(1):1–44, 1960.

[5] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31:1268–1287, 1988.

[6] S. Dencheva, C. R. Prause, and W. Prinz. Dynamic self-moderation in a corporate wiki to improve participation and contribution quality. In *Eur. Conf. Comp.-Supp. Coop. Work*, ECSCW. Springer, 2011.

[7] A. Forward and T. C. Lethbridge. The relevance of software documentation, tools and technologies: a survey. In *DocEng*, pages 26–33. ACM, 2002.

[8] G. Hardin. The tragedy of the commons. *Science*, 162:1243–1248, 1968.

[9] M. Harman. Why source code analysis and manipulation will always be important. In *SCAM*. IEEE, 2010.

[10] B. Hoisl, W. Aigner, and S. Miksch. Social rewarding in wiki systems — motivating the community. In *Online Communities and Soc. Comp.* Springer, 2007.

[11] A. Hunt and D. Thomas. *The Pragmatic Programmer*. Addison-Wesley Longman, 1999.

[12] A. Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, 2007.

[13] W. E. Lewis. *Software Testing and Continuous Quality Improvement*. Auerbach, 2005.

[14] B. P. Lientz and E. B. Swanson. Problems in application software maintenance. *Communications of the ACM*, 24(11):763–769, 1981.

[15] J. S. Poulin. Populating software repositories: Incentives and domain-specific software. *Journal of Systems and Software*, 30:187–199, 1995.

[16] C. Prause, J. Kuck, S. Apelt, R. Oppermann, and A. B. Cremers. Interconnecting documentation - harnessing the different powers of current documentation tools in software development. In *Ninth ICEIS*, volume ISAS, pages 63–68. INSTICC, 2007.

[17] C. R. Prause. Maintaining fine-grained code metadata regardless of moving, copying and merging. In *9th International Working Conference on Source Code Analysis and Manipulation*, SCAM. IEEE, 2009.

[18] C. R. Prause. A software project perspective on the fitness and evolvability of personal learning environments. In *Exploring the Fitness and Evolvability of Personal Learning Environments*, 2011.

[19] C. R. Prause and S. Apelt. An approach for continuous inspection of source code. In *International Workshop on Software quality*, WoSQ. ACM, 2008.

[20] C. R. Prause and M. Eisenhauer. Social aspects of a continuous inspection platform for software source code. In *Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE. ACM, 2008.

[21] C. R. Prause, M. Jentsch, and M. Eisenhauer. Mica - a mobile support system for warehouse workers. *International Journal of Handheld Computing Research (IJHCR)*, 2(1):1–24, January-March 2011.

[22] C. R. Prause, R. Reiners, and S. Dencheva. Empirical study of tool support in highly distributed research projects. In *5th International Conference on Global Software Engineering*, ICGSE, pages 23–32. IEEE Computer Society, 2010.

[23] C. R. Prause, R. Reiners, S. Dencheva, and A. Zimmermann. Incentives for maintaining high-quality source code. In *Psychology of Programming Work in Progress Meeting*, PPIG WiP, 2010.

[24] J. Raskin. Comments are more important than code. *ACM Queue*, 3(2):64–62 (sic!), 2005.

[25] B. Selic. Agile documentation, anyone? *IEEE Software*, 26(6):11–12, Nov/Dec 2009.

[26] D. Spinellis. Code documentation. *IEEE Software*, 27:18–19, 2010.

[27] R. D. Stacey. *Strategic Management and Organisational Dynamics: the challenge of complexity*. Financial Times, 1999.

[28] M. Umarji and F. Shull. Measuring developers: Aligning perspectives and other best practices. *IEEE Software*, 26(6):92–94, Nov./Dec. 2009.

[29] L. Williams and B. Kessler. The effects of "pair-pressure" and "pair-learning" on software engineering education. In *13th Conference on Software Engineering Education & Training*. IEEE, 2000.

[30] S. Wray. How pair programming really works. *IEEE Software*, 27:50–55, 2009.

[31] T. Yamabe, V. Lehdonvirta, H. Ito, H. Soma, H. Kimura, and T. Nakajima. Applying pervasive technologies to create economic incentives that alter consumer behavior. In *UbiComp*. ACM, 2009.