2015 IEEE 10th International Conference on Global Software Engineering

Distributed Software Engineering in Collaborative Research Projects

Michael Derntl, Dominik Renzel, Petru Nicolaescu, István Koren, Ralf Klamma
RWTH Aachen University
Advanced Community Information Systems (ACIS)
Informatik 5, Ahornstr. 55, 52056 Aachen, Germany
Email: *lastname*@dbis.rwth-aachen.de

***Abstract*—Collaborative research projects involve distributed construction of software prototypes as part of the project methodology. A major challenge thereby is the need to establish a developer community that shall effectively and efficiently align development efforts with requirements offered by researchers and other stakeholders. These projects are inherently different in nature compared to commercial software projects. The literature offers little research on this aspect of software engineering. In this paper, we outline the challenges in this context and present a methodology for distributed software engineering in collaborative research projects. The methodology covers all major aspects of the software engineering process including requirements engineering, architecture, issue tracking, and social aspects of developer community building in collaborative projects. The methodology can be tailored to different project contexts and may provide support in planning software engineering work in future projects.**

Keywords*—Distributed software engineering, Collaborative research projects, Open source software, Requirements engineering, Development infrastructure, Continuous integration, Methodology

I. Introduction and Motivation

Over the past eight years, the European Commission has spent up to 1.5 billion Euros annually on funding collaborative research and innovation in Information and Communication Technologies under the umbrella of the Seventh Framework Programme for Research (FP7) [1]. Likewise, the National Science Foundation (NSF) has requested almost one billion US Dollars for the the CISE (Computer & Information Science & Engineering) program alone for fiscal year 2016 [2]. These figures demonstrate that massive amounts of public funds are spent on research projects with an information technology (IT) focus. One key activity in such projects, whether explicit or not, is software engineering. It may serve different purposes in different projects, e.g. producing prototypes for innovative IT applications or provision of computing infrastructure. Regardless of the purpose, spending the available funds effectively and efficiently can be challenging.

Research projects in science and engineering have thus become a topic of rising interest in software engineering research lately [3]. There is awareness that the use of sound software engineering methods and principles is pivotal in producing software that can be used and maintained for scientific purposes as well as for driving innovation. Yet, many people involved in software development processes in science lack formal training in software engineering, and software engineering in research projects comes with inherently different characteristics compared to, for instance, software engineering in commercial IT projects [3]. While commercial IT projects ultimately strive for financial success and customer satisfaction, which are both easily measurable, research projects strive for scientific success in terms of reputation and impact through high-profile publications in prestigious outlets. In research projects the produced software is often simply an instrument that is required to conduct research. Therefore, software artifacts output by research projects are often prototypes—regarded as boundary objects of innovative technology and scenarios [4], [5]—which typically cannot benefit from a well staffed work force to reach the maturity of commercial products. Also these software artifacts are not necessarily part of the promised project output. Moreover, although research projects typically follow agreed scientific methodologies, each is unique, aiming to explore and discover unknown territory from the baseline. Such "once-only projects" expose a significant risk of failing [6]. Hence, measures have to be taken to establish effective and efficient software engineering practice and ensure the quality and sustainability of the software outputs.

Fueled by the rise of Web based information and communication technologies (ICT), the internationalization and distribution of teams and workers in research and development has increased tremendously in the last two decades. This has led to challenges related to increasing costs, increased travel frequency, uncertainty, cultural issues, management challenges and the need for improved coordination tools in software engineering [7], [8]. In projects funded under FP7, for instance, it has been mandatory that different partners from at least three different countries participate in a collaborative project [9]. In such projects, which typically have a fixed duration of two to four years, there are limited time and resources to adopt, install and apply an agreed development methodology within the consortium. In the literature there is extensive research into these distributed, virtual teams in an R&D project context. A contemporary overview can be found in [10], where the authors conclude that further research is needed into supporting infrastructures for distributed R&D teams, since all teams must find ways and means for sharing knowledge and managing their work. It is therefore essential to facilitate development work with various measures of consortium and stakeholder involvement, including decision making instruments, commitment to management of the software process, and clear communication structures (e.g. [11]). The methodology presented in this paper tries to achieve exactly that. In unfolding the methodology, we borrow relevant ideas from research on global software engineering, in particular collaborative development environments [12] and the tools to support it like version control, issue trackers, integration, and communication tools [13].

978-1-4799-8409-1/15 $31.00 © 2015 IEEE
DOI 10.1109/ICGSE.2015.12

Another issue of increasing relevance is the licensing of products created using public funds. Public funding agencies often endorse the use and development of Open Source Software (OSS). For instance, the European Commission sees the open source development model as a "very effective way to collaboratively develop software with fast take-up and improvement cycles, [...] a vehicle for the dissemination of results of ICT research projects" [14]. The methodology in this paper relies heavily on the open source development model.

To preserve and spread good practice in this context we present successful solutions deployed and refined over the last years in the form of a methodology that can be reused in other research projects with comparable scope and challenges. The paper focuses on collaborative projects that involve distributed partners and require distributed development, integration, deployment and management of software based on state-of-the-art research and technologies, end-user involvement, and informed decision making. Examples include research projects that build innovative integrated information systems, or ones that build a common middleware layer for scientific applications—see [15] for an example. The roots of the insights presented in this paper lie in the technical leadership of two inherently different, large-scale research projects in FP7's thematic area "information and communication technologies," namely ROLE [http://role-project.org] and Layers [http://learning-layers.eu], which are presented in more detail later on. While in ROLE the task was to develop and deploy a platform for responsive open learning environments supporting self-regulated learning [16], the challenge in Layers has been to facilitate the development and fast deployment of a scalable, flexible infrastructure for mobile social apps at the workplace [5]. There is negligible overlap in functional requirements in these two projects, yet both exposed a considerable number of common challenges and non-functional requirements related to the software architecture and the engineering & integration processes. We argue that many of these challenges are highly relevant to large-scale research projects that involve distributed software development.

II. Context and Challenges

A. Empirical Project Context

The empirical context underlying this paper stems from two large scale integrated projects funded by the European Commission under the ICT work programme in FP7. Both projects were collaborative projects, with ambitious goals and a large consortium of partners, requiring dozens of people with different backgrounds and affiliations to collaboratively perform cutting edge research and development. The basic facts of these two projects are outlined below.

ROLE Project. Responsive Open Learning Environments (ROLE) was a large scale integrated project running from 2009 to 2013. Sixteen partners from Europe and China were part of the consortium, and the total budget amounted to roughly 8.5 million Euros, of which a considerable share was spent on software engineering related activities. The core work packages are R&D activities developing a widget-based platform for self-regulated informal learning in personal and group learning environments. The authors' department was technical coordinator of the project, responsible for the complete software lifecycle. While all partners participated in the software engineering process in various roles, more than half of the partners were technical partners participating in active OSS development. Other partners were primarily focused on theoretical research, requirements engineering, piloting and exploitation activities. ROLE did not include a dedicated work package dealing with the software process, architecture, and infrastructure.

Layers Project. Layers is a four-year project which was kicked off in 2012. Eighteen partners are part of the consortium and the total budget amounts to roughly 12.5 million Euros. The core work packages are R&D activities developing innovative technical infrastructure and software applications for workplace learning scenarios. All partners participate in the software engineering process in various roles, and half of all consortium partners are partners that actively participate in the production and/or management of computing infrastructure, source code, and binaries. The other partners are primarily focusing on research and piloting activities. There is a dedicated work package dealing with the software process, architecture and infrastructure in the project, which is being led by the authors of this paper. In this project several of the instruments previously deployed in ROLE were adopted and extended.

B. Common Challenges

To set the context of interest for the methodology presented in this paper, we outline in the following a list of non-functional challenges for distributed software engineering in collaborative research projects, drawn from the experiences in the two projects mentioned above. In the following Section III, we then show how the presented methodology provides instruments and activities to tackle these challenges.

Decision making. Architectural decisions with considerable scope need to be made early in the project, and it is well known that the early mistakes in a project are the costliest [17]. Decisions should thus be made transparently and traceably.

Short cycles. In the time frame of an research project, development cycles must be kept short, and initial architectures and prototypes must be provided early to allow for frequent refinement loops driven by research and end-user involvement.

Impact. Typically, exploitation and impact of project results are key performance indicators for funding agencies, while in reality the market-readiness of ICT research project outcomes is often very limited (e.g. [18]).

Distributed team. The developer community is distributed across partners, often across borders as well, and needs to swiftly grow and flourish.

Support tools. The collaborative development environment and process, including procedures and tools for source code management, issue tracking, software branding, and similar, need to be defined and implemented to smoothly align with the scope and dynamics of the development efforts [13].

OSS Licensing. An OSS strategy usually helps to sustain and transfer project results into practice and is thus interesting for funding agencies [14]. In large project consortia, agreement on one license for all outputs is hard to reach, particularly because partners often bring in software components that already have a license attached.

Stakeholder commitment. Since there is the danger of a lack of stakeholder commitment to project activities, software process instruments need to be highly inclusive and supportive.

Baseline. In research projects, the state of the art builds the baseline for research and innovation. 'Quick & dirty' approaches may thwart ambitions for cutting-edge research.

Unknown territory. Research projects break fresh ground, so the outcomes remain unspecified beforehand; this poses challenges for the software engineering activities since these need to synchronize well with the primary research activities.

Ambitions. Research projects aim at scientific success in terms of innovation, reputation and impact by high-profile publications. Conflicting ambitions of collaborating institutions can pose obstacles for rational and efficient decision making.

III. Proposed Methodology

Meeting the challenges laid out in the previous section requires a sound methodological basis. The deployed methodology relies on three main activity threads, which are described in the following sub-sections: Convergence (Section III-A), Stakeholder Engagement (Section III-B), and Software Development (Section III-C). An overview matrix pitching instruments of the methodology presented in this section (grouped by tool-related and people-related instruments) to the challenges posed in Section II is displayed in Table I. It shows that each challenge is addressed by multiple instruments, and that each instrument supports multiple challenges.

TABLE I. Challenges (rows) addressed by the instruments of the methodology (columns)

		Instruments									
		Tools							**People**		
		Technology Survey	Requirements Bazaar	House of Quality	Continuous Integration	Issue Tracking	Source Code Repository	Developer Hub	Developer Task Force	Architecture Board	Stakeholder Engagement
Challenges	Decision making	❍	❍	❍				❍	❍		❍
	Short cycles		❍		●	●	❍	❍	❍		❍
	Impact	❍	❍	❍		❍	❍	❍		❍	❍
	Distributed team		❍		❍	❍	●	●	●		
	Support tools								❍	❍	
	OSS licensing	❍					❍	❍	❍	❍	
	Commitment		●	❍	❍	❍			❍	❍	●
	Baseline	●		❍							
	Unknown territory	❍	❍		❍	❍			❍		
	Ambitions	❍	❍	●					❍	●	❍

Legend: ❍ *support* ● *strong support*

A. Convergence Activities and Instruments

Technology Survey. To obtain an overview of relevant existing technologies, an established activity is to conduct a desktop search, which we called *technology survey*. Existing technologies and options for the present development cycle are surveyed by the technical partners and documented internally. This activity tries to ensure that the relevant baseline approaches and technologies are explored and assessed. In early cycles this activity will help to remedy challenges related to architectural decision making. Also, such a survey will help to reveal whether licensing models of adopted components and software are compatible with the project's licensing model.

Requirements Engineering. Diversity of end-user and test-bed characteristics calls for heavy stakeholder involvement during all stages of the project. It is well known that requirements negotiation and continuous adaptation are central goals of any information system development endeavor (e.g. [19]). As a recent survey of requirements engineering tools has shown [20], most tools cover well the requirements elicitation and specification aspects, but the survey identified a considerable margin for improvement of requirements management. As a consequence, non-developers are more likely to avoid participation in essential requirements engineering tasks. This dilemma is tackled by Social Requirements Engineering [21] by employing concepts of social software (e.g. commenting, voting, communication and sharing artifacts) and combines them in a portal for end-user communities. In the *Requirements Bazaar* [http://requirements-bazaar.org] [21] Web application end-users may engage in an informal requirements elicitation process by contributing requirements and enriching them with user stories and images.

Technology Assessment. Projects need to be able to continuously integrate new components, tools and updates based on new insights, technological advances, and changing requirements. While doing this, it is essential to continuously make sound architectural decisions informed by the actual needs—both functional and non-functional—of all stakeholders. A technique that facilitates this is *Quality Function Deployment (QFD)* [22], and the QFD instrument we suggest to adopt for mapping features and requirements is called *House of Quality* (HoQ) [23]. This instrument has been adopted successfully by many large companies, including Ford, Xerox and AT&T for their product development activities. The HoQ can be used to match the list of weighted functional requirements obtained with features offered by (a subset of) the baseline technologies that establish the integrated system. This supports the assessment of existing systems in terms of how well they perform in meeting user requirements, which finally leads to informed architectural decisions. One of the main insights offered by a HoQ is an understanding of the importance of each feature for project success. For constructing a HoQ, the list of user requirements from the social requirements engineering process can be used as a starting point. The most important part of the HoQ methodology is setting the requirements in relation with product features, which is achieved using a number system that directly impacts the resulting software.

B. Stakeholder Engagement

As stated above, large research projects aim to meet the aims of various stakeholder groups. Put simply, researchers aspire high-quality publications, developers target an efficient development process, and application partners wish for software that is readily usable in their context. It is therefore an important task to provide platforms to engage stakeholders and allow them to communicate with each other. Various approaches for setting up these platforms can be employed, amongst them co-design teams consisting of representatives from end-user partners, researchers, and developers. These are permanent, open work groups within the project that work on certain design ideas to align the design of the prototypes with the requirements of the application partners. Their main concern is to drive technological innovation based on real problems and authentic scenarios. Since there is often a wide variety of different ideas, it is of high importance to find the commonalities and key elements across these ideas. Bringing together the various feature requests is a major challenge; assessment instruments like the House of Quality facilitate this process. A constant dialogue between co-design teams and developers is maintained by having at least one team member that is also a representative of the developer task force. Endeavors of co-design teams can then be directly transferred to developer task force meetings and vice versa.

C. Software Development

In [7], the authors introduce four principal concepts of organizing virtual R&D teams ranging from decentralized self-coordination to centralized venture team with increasing degree of centralized control. Taking into account the motivational situation and the management structures at research centers and universities typically involved in collaborative research projects, we argue for a blended approach combining decentralized self-coordination for the short- and mid-term agenda, with a system architecture core team installed to deal with long-term, high-impact decisions and strategic development objectives. For the decentralized self-coordination we propose a *developer task force* and for the core architecture control we propose a governing body called *architecture board.*

Developer Task Force. Splitting the software development effort in a research project into artificial teams should be avoided [15]. To facilitate the emergence of a team spirit with shared ownership the author recommends to build a single virtual team. In this regard, and in line with our previous research project experiences, we suggest to establish a *developer task force* as an informal community to bundle developer resources distributed among project partners in a virtual, loosely coupled team structure. Typically, the developers in a research project are either professional developers from participating companies or researchers/PhD students at research partner institutions. A unified, virtual task force shall help to sustain a healthy "steady heartbeat" [6] during the software engineering process in the face of these challenges. The members can influence decisions such as the software engineering methods employed (e.g. agile methods, incremental, iterative, etc.) for the project, development infrastructure, tools used for continuous integration, issue tracking, code management and other such collaboration tools. The task force can be seen as an exploration unit with clear competences, and it shall be self-managed and given considerable autonomy for defining and achieving the short- and mid-term objectives. This will lower the pressure coming from rules and regulations, which is known to be a failure factor in software projects [6].

Architecture Board. A governing body, which we chose to label the "architecture board" to reflect its duty of making and enforcing global decisions, shall be established as an authority to make binding decisions for all partners. It has been suggested [15] that an authority should enforce the policies and be the only one allowed to change policies. The architecture board will also help to increase stakeholder commitment due to stronger involvement in decision making. Informed, consensus-oriented decision making—optimally supported by consulting instruments like House of Quality—might also increase the chance of post-project exploitation and impact of results, which is a key issue of concern for funding agencies.

Developer Hub. In order to sustain the produced software artifacts beyond the scope of the research project, these should be developed and released using state-of-the-art practices and tools from the OSS community [15]. The developer task force shall therefore set up a *developer hub* on the Web, acting as an information center for all developer related resources in the project targeting internal and external members. This includes, for instance, apps and nightly builds for download, API documentation, installation instructions, links to all developer tools, and so forth. This will also facilitate the establishment and maintenance of knowledge about project-specific information items—e.g. technological agreements, process agreements, build and deployment related information—that are known to be highly important for satisfying the information needs of distributed software engineers [24]. Since documentation is a key feature of the software project landscape [25], a well-maintained developer hub will also facilitate coping, e.g. with new developers joining. We propose the following essential tools for the developer hub:

- *Social Requirements Engineering* [21]: we recommend the use of Requirements Bazaar as an open community toolkit that allows requirements generation and prioritization. This is done in a bazaar-like metaphor, negotiating around requirements using posting of artifacts, comments, and votes. All co-design activities in the project use it as a boundary object for sharing and negotiating requirements with the developer task force. Once a developer commits to the realization of a requirement, it is pushed into the issue tracker.
- *Issue Tracking*: allows the management of software issues, including bugs, feature requests, etc. Most issue tracking software packages offer more than pure issue listings. Based on experience we prefer JIRA by Atlassian, offering free licenses to OSS projects [https://www.atlassian.com/opensource]. JIRA is also used by major OSS organizations such as the Apache Foundation. We deployed a two-way integration of JIRA with Requirements Bazaar (cf. [26]).
- *Source Code Integration*: Build-level code integration tools support and access source code management systems, e.g. to obtain code for regular integrated builds. In our projects we prefer the open source solution Jenkins [http://jenkins-ci.org]. Jenkins can be used to run automated tests and nightly builds, driven by the source code repository. Regular builds offer timely notifications for the people in charge of integration.
- *Source Code Repository*: In a distributed software engineering process it is key to have a reliable source code management in place. There are freely available solutions available. We have chosen to use Git, since it offers sophisticated support for distributed development (e.g. full-fledged local repositories) and offers desktop and Web clients (e.g. GitHub) that integrate well with most state-of-the-art software management tools.

- *Documentation*: Finally, the developer hub provides comprehensive documentation of all software and tools offered and developed in the project.

IV. Conclusion

In this paper we have proposed a methodology for software engineering in software-intensive, collaborative research projects. We claimed that these projects often expose common challenges regarding the software engineering process. We presented activities and instruments that allow approaching these challenges with proven tools and established practices from real-world developer communities. Typical collaborative research projects involve goals and strategies of researchers, application partners, and developers. The key to let convergence win over divergence is the set of instruments deployed to support continuous integration at all levels and in all areas where stakeholder interests meet. In particular we proposed the developer hub approach, which acts as a center offering tools and practices facilitating the developer task force.

While the methodology presented here stems from projects involving a large number of partners, it appears reasonable and applicable also for smaller distributed projects. In fact, we believe that the complexity of the software to be developed plays a more important role for the necessity of deploying certain support instruments than the mere size of the developer community. With a smaller number of partners some people-related instruments (e.g. the architecture board) should become simpler to manage. The tool-related instruments (e.g. the Requirements Bazaar or the issue tracker) will also prove useful in non-distributed or single-developer projects.

With this contribution we aim to preserve previously and currently successful practice in a way that allows future project consortia to plan and execute their software engineering processes by tailoring the presented methodology to their needs. Last but not least we aim to establish a professional culture of sharing and continued refinement of software engineering good practices in research projects.

Acknowledgment

This research was supported by the European Commission in the Seventh Framework Programme project "Layers", grant no. 318209. This paper reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

References

[1] European Commission, "Budget – FP7 – Research," http://ec.europa.eu/research/fp7/index_en.cfm?pg=budget, 2013.

[2] National Science Foundation, "FY 2016 NSF Budget Request to Congress," http://www.nsf.gov/about/budget/fy2016/pdf/18_fy2016.pdf, 2015.

[3] J. C. Carver, "First International Workshop on Software Engineering for Computational Science & Engineering," *Computing in Science & Engineering*, vol. 11, no. 2, pp. 7–11, 2009.

[4] H. Rhinow, E. Koeppen, and C. Meinel, "Prototypes as Boundary Objects in Innovation Processes," in *Design Research Society 2012: Bangkok. Conference Proceedings*, vol. 4. DRS, 2012, pp. 1581–1590.

[5] T. Ley, J. Cook, S. Dennerlein, M. Kravcik, C. Kunzmann, K. Pata, J. Purma, J. Sandars, P. Santos, A. Schmidt, M. Al-Smadi, and C. Trattner, "Scaling informal learning at the workplace: A model and four designs from a large-scale design-based research effort," *British Journal of Educational Technology*, vol. 45, no. 6, pp. 1036–1048, 2014.

[6] H. Huijgens, R. van Solingen, and A. van Deursen, "How to build a good practice software project portfolio?" in *36th Int. Conf. on Software Engineering, Companion Proceedings*. ACM, 2014, pp. 64–73.

[7] O. Gassmann and M. v. Zedtwitz, "Trends and determinants of managing virtual R&D teams," *R&D Management*, vol. 33, no. 3, pp. 243–262, 2003.

[8] S. Beecham, P. O'Leary, I. Richardson, S. Baker, and J. Noll, "Who are we doing Global Software Engineering research for?" in *Proc. 8th IEEE Int. Conf. on Global Software Engineering*. IEEE, 2013, pp. 41–50.

[9] European Commission, "Regulation (EC) No 1906/2006 of the European Parliament and of the Council. Official Journal of the European Union, L 391/1." http://ec.europa.eu/research/participants/data/ref/fp7/90749/ecrulesforparticipation_en.pdf, 2006.

[10] N. A. Ebrahim, S. Ahmed, and Z. Taha, "Establishing Virtual R&D Teams: Obliged Policy. CoRR, abs/1208.0994," http://arxiv.org/abs/1208.0944, 2012.

[11] H. Berger and P. Beynon-Davies, "The utility of rapid application development in large-scale, complex projects," *Information Systems Journal*, vol. 19, no. 6, pp. 549–570, 2009.

[12] G. Booch and A. W. Brown, "Collaborative development environments," *Advances in Computers*, vol. 59, pp. 1–27, 2003.

[13] F. Lanubile, C. Ebert, R. Prikladnicki, and A. Vizcaino, "Collaboration tools for global software engineering," *IEEE Software*, vol. 27, no. 2, pp. 52–55, 2010.

[14] European Comission, "Free and open source software activities in European Information Society initiatives," http://cordis.europa.eu/fp7/ict/ssai/foss-home_en.html, no date.

[15] P. Kunszt, "Grid Middleware Development in Large International Projects – Experience and Recommendations," in *Int. Conf. on Software Engineering Advances*. IEEE, 2007, pp. 82–86.

[16] R. Carneiro, P. Lefrere, K. Steffens, and J. Underwood, Eds., *Self-regulated Learning in Technology Enhanced Learning Environments: A European Perspective*. Rotterdam: SensePublishers, 2011.

[17] J. C. Westland, "The cost of errors in software development: evidence from industry," *The Journal of Systems and Software*, vol. 62, no. 1, pp. 1–9, 2002.

[18] European Comission, "FP6 IST Impact Analysis Study: Final Report," http://cordis.europa.eu/fp7/ict/impact/documents/wing-pilot-fp6-final-report-18-12-09.pdf, 2009.

[19] D. P. Truex, R. Baskerville, and H. Klein, "Growing systems in emergent organizations," *Communications of the ACM*, vol. 42, no. 8, pp. 117–123, 1999.

[20] J. M. Carillo de Gea, J. Nicolas, J. L. Fernández Alemán, A. Toval, C. Ebert, and A. Vizcaíno, "Requirements Engineering Tools," *IEEE Software*, vol. 28, no. 4, pp. 86–91, 2010.

[21] E. L.-C. Law, A. Chatterjee, D. Renzel, and R. Klamma, "The Social Requirements Engineering (SRE) Approach to Developing a Large-Scale Personal Learning Environment Infrastructure," in *EC-TEL 2012, LNCS 7583*, A. Ravenscroft, S. Lindstaedt, C. D. Kloos, and D. Hernández-Leo, Eds. Springer Verlag, 2012, pp. 194–207.

[22] Y. Akao, *Quality Function Deployment: Integrating Customer Requirements into Product Design*. Cambridge: Productivity Press, 1990.

[23] J. R. Hauser and D. Clausing, "The House of Quality," *Harvard Business Review*, vol. 66, no. 3, pp. 63–73, 1988.

[24] K. Dullemond and B. van Gameren, "What distributed software teams need to know and when: an empirical study," in *Proc. 8th IEEE Int. Conf. on Global Software Engineering*. IEEE, 2013, pp. 61–70.

[25] B. Dagenais, H. Ossher, R. K. E. Bellamy, M. P. Robillard, and J. P. de Vries, "Moving into a new software project landscape," in *Proc. 32nd ACM/IEEE Int. Conf. on Software Engineering - Volume 1*. ACM, 2010, pp. 275–284.

[26] D. Renzel, M. Behrendt, R. Klamma, and M. Jarke, "Requirements Bazaar: Social Requirements Engineering for Community-Driven Innovation," in *21st IEEE Int. Requirements Engineering Conference (RE)*. IEEE, 2013, pp. 326–327.