Preparing Research Projects for Sustainable Software Engineering in Society

Dominik Renzel, István Koren, Ralf Klamma, Matthias Jarke
RWTH Aachen University
Advanced Community Information Systems (ACIS)
Informatik 5, Ahornstr. 55, 52056 Aachen, Germany
Email: renzel,koren,klamma,jarke@dbis.rwth-aachen.de

***Abstract*—With the pervasive need for digitization in modern information society, publicly funded research projects increasingly focus on engineering digital approaches to manage societal processes. Such projects inherently face the challenge of establishing a sustainable software engineering culture. A major challenge thereby is that project consortia need to establish a distributed developer community that effectively and resource-efficiently aligns development efforts with the goals and needs of complex societal constellations beyond project lifetime. In this paper we extract empirical evidence from longitudinal studies in two large-scale research projects to outline typical challenges in such problem contexts and to develop an open source software engineering methodology for research projects, including supportive infrastructure and social instruments of community building and awareness. We thus contribute a comprehensive strategy preparing collaborative research projects for sustainable societal software engineering.**

I. Introduction

With the pervasive need for digitization in information societies, we find strong competition of commercial as well as publicly funded research projects increasingly focusing on engineering digital approaches to manage all kinds of societal processes. Indeed, many research challenges (e.g. fusion research, etc.) are so complex that they can only be addressed at a societal level [1].

On the part of industry, a vast majority of CEOs (89%) acknowledges that commitment to sustainability translates into real impact in their industry [2], but still faces challenges towards achieving [3]. For the software industry, sustainability is at most a non-functional requirement costly to fulfill and thus only pursued if the company has sufficiently stable stance to argue for it or the customer explicitly requires it [3]. Major software companies have installed own R&D strategies and business models to collaborate on new interoperability standards, to gain trust in their infrastructure and to attract developers to use their platforms and APIs for building, growing, and sustaining ecosystems of mobile/Web applications.

For academia and SMEs, the situation is different. Major funding agencies like the European Commission state sustainability of project results beyond project lifetime as fundamental requirement for overall success. Public sector funding agencies are strongly aware of societal challenges, in particular sustainability, in much larger diversity than in profit-oriented enterprises. Over the past eight years, the European Commission has spent over 50 billion Euro on funding collaborative research and innovation in ICT under the umbrella of the Seventh Framework Programme for Research (FP7) [4]. Likewise, the NSF requested almost one billion USD for 2014 in the *Computer & Information Science & Engineering (CISE)* program alone [5]. These figures demonstrate that massive amounts of public funds are spent on research projects with IT focus. One key activity in such projects is software engineering for different purposes, e.g. producing prototypes for innovative IT applications or provision of large-scale computing infrastructure. Especially larger integrating projects produce technological artifacts such as infrastructures, frameworks, and specifications as basis for growing ecosystems of derivative software products. The European Commission requires research plan proposals to explicitly address sustainability as goal in terms of how well project results and development processes are prepared to survive beyond project termination. With the end of project funding, this goal becomes challenging, as consortia split up and partners turn their ways in terms of new emerging goals and funding opportunities. Achieving such sustainability must thus be clearly addressed during project lifetime.

Especially in the ICT calls, *Open Source Software (OSS)* is seen as central means for sustainable software development. Clearly, this stance focuses on societal and economic concerns as the primary purposes of public research projects. While the OSS movement started as a niche phenomenon, it is now omnipresent in virtually all sectors of modern information societies. The European Commission [6] and the US government [7] issue strategies to make reasonable use of and actively contribute resources to OSS development. Collaborative research projects are an important source of such contributions, as their organization strongly resembles the inner workings of OSS communities. *"With a large number of projects delivering Open Source software we should not miss the opportunity to share experience and know-how on [...] how to make project results sustainable over time [...]"* [8].

However, commercial software providers face a certain inability to practice sustainability design within software engineering due to the lack of methodology and tool support [3]. In this paper, we argue that this finding is even more true for research projects due to their different structure, motivations, and goals. In particular, we emphasize the importance of preparing projects for sustainability of project results and software engineering processes. Research projects can reach this goal with the help of an agile methodology inspired by established DevOps and OSS approaches, supported by a combination of technical and social instruments to establish, maintain, adapt, and sustain software engineering beyond project lifetime. We observe that DevOps methodologies generally do not reflect contributions of end-user communities, although these play important roles in many phases of both commercial and OSS development. Thus, engineering software for societies must explicitly include the role of end-users in participatory processes from planning over design to evaluation to gain their trust and confidence in the built software artifacts. Our work is thus mostly located in the intersection of social and technical aspects regarding sustainable software engineering in societies (cf. Karlskrona Manifesto [9]). To preserve and spread effective practices in this context we studied challenges and successful solutions in several research projects over the last years and synthesized our findings in a software engineering methodology and related infrastructure designed for reuse in other research projects with comparable scope.

The rest of this paper is structured as follows. In Section II we discuss existing literature on distributed software engineering in research and development contexts. In Section III we derive specific challenges in research projects from two case studies of EU-funded large-scale integrating projects. In Section IV we present our DevOpsUse methodology and infrastructure for sustainable societal software engineering. In Section V we report the key lessons learned from the empirical evidence in our case studies. In Section VI, we conclude this paper with a discussion of limitations and future measures.

II. Related Work

Research projects in science and engineering have recently become a topic of rising interest in software engineering research [10]. The use of sound software engineering methods and principles is pivotal in producing software for scientific and innovative purposes. Yet, academic staff involved in software development often lack training in software engineering [11]. Software engineering in research projects comes with inherently different characteristics compared to commercial IT projects [12]. With special regard to sustainability, the majority of projects differ in terms of goals, nature of delivered products and internal processes. Although companies reflect an understanding of sustainability in quality criteria such as usability, maintainability, or agility to update [3], top level goals of software project management still boil down to staying within time and budget, to achieve customer satisfaction, and finally to increase sales and profit. In contrast, research projects strive for scientific success in terms of reputation and impact in prestigious publications. Often, software is merely considered a research instrument, while novelty, feasibility of research methods and results as proof of concept are more important. Thus, software artifacts designed in research projects frequently stay prototypes, regarded as boundary objects of innovative technology and scenarios [13]. Such prototypes require sustained development to reach commercial maturity, as they usually lack commercial success characteristics. Although research projects typically follow agreed scientific methodologies, each is unique since it aims to explore and discover unknown territory starting from the state-of-the-art. Such "once-only projects" expose a significant risk of failing [14]. To mitigate this risk, research projects must establish effective and efficient software engineering practice among multiple distributed developer teams early to prepare for quality and sustainability of both development process and products. Extensive research into distributed teams in R&D projects has revealed the concrete need for supporting methodologies and infrastructures [15]. Software engineering must be coordinated with various instruments creating sustained consortium and stakeholder involvement, including decision making, development, and communication [16].

III. Empirical Contexts and Challenges

In this section, we discuss two longitudinal case studies in large-scale integrated research projects funded by the European Commission. Both projects set ambitious development goals towards societal transformation and involved broad consortia of partners with different backgrounds and motivations. All projects developed enabling technologies in terms of frameworks, infrastructures, or technology standards as basis for growing ecosystems of societal software. With societal software we define software that pays as much attention to the process of creating the software product as to the software product itself in analogy to participatory negotiation and design processes in society, e.g. large infrastructure projects.

Project ROLE

Responsive Open Learning Environments (ROLE) was a EU FP7 large-scale integrating project from 2009 to 2013, involving sixteen partners from Europe and China with a budget of 8.5 Mio Euro. The objective of the ROLE project was to develop a responsive and open widget-based platform for self-regulated informal learning in *personal learning environments (PLE)*. With the ROLE SDK, the project delivered a reference implementation of the platform including widget developer tools. End-users with corporate and institutional learning background in five testbeds were involved in *requirements engineering (RE)*, co-design and evaluation activities.

During the first months, the consortium produced a *technology survey* as dedicated deliverable with the goal to outline the current state-of-the-art in personal learning technology popular in the testbeds. Finding a common baseline among many diverse technologies required intense partner negotiation. In parallel, partners conducted face-to-face workshops and focus-group interviews as part of RE. Resulting non-digital artifacts were digitized and documented in a project deliverable. Already in early stages, face-to-face RE activities turned out as excessively resource-intensive and thus unscalable for frequent reiteration. The consortium thus started research into remote end-user participation in a negotiation-driven social requirements engineering process [17].

With the beginning of development, the technical lead established open biweekly developer meetings, gathering representatives from all technical partners. This *developer task force (DTF)* enjoyed considerable freedom in short-term development. Development goals were governed by the input from requirements engineers. However, with each partner reporting progress and issues, meetings became increasingly longer. The DTF addressed this challenge with issue-centric meetings, supported by an issue tracker. Every agenda item had to be reflected by a corresponding issue. Discussion items were immediately documented in the respective issues. Meeting time could be drastically reduced with all issues documented traceably. However, cases of disagreement and long-term decisions were escalated to project management. An intermediate senior technical board would have been preferable.

From the start, the consortium agreed on producing OSS. The DTF thus discussed OSS *licensing* early in the first year with the goal to agree on one license model fitting all partners' exploitation strategies. Legal departments of the partners then issued recommendations, which effectively rendered a single-license model impossible. The consortium then had to spend considerable resources in reaching consensus on a set of compatible licenses. Additionally, the DTF agreed on SourceForge as public platform for code hosting and revision control. Retrospectively, SourceForge turned out as wrong choice for two reasons. First, missing support for managing smaller repositories under the same organization quickly led to a monolithic code base of unhandleable size, including code for the core platform, services, widgets, and experiments. Second, GitHub quickly outgrew and displaced SourceForge, becoming the de-facto standard platform for highly visible OSS projects. Later migration to GitHub was not successful due to resource shortage for a major cleanup.

In first development stages, individual member institutions of the DTF developed prototypes of a widget-based PLE infrastructure or learning widgets in isolation, however still adhering to the same open standard widget specifications. In later stages, the need for providing the ROLE SDK as bundled product raised the issue of integrating the many components. The DTF addressed this challenge with a Hudson (now Jenkins) *continuous integration (CI)* system for nightly builds and automated testing. With the availability of a CI system, the project experienced a boost in convergence.

In later project stages with tangible results available, DTF members targeted external OSS conventions such as FOSDEM or Apache meetings to disseminate results to wider audiences, including other research projects. Additionally, the DTF organized own coding competitions and hackathons [18], piloting with internal developers only, and later addressing external developers. Such events were door openers to establishing valuable contacts across projects, sharing common best practice, raising awareness, and sustaining project results as contributions to other OSS projects. The interaction with external parties should start with the availability of development results.

Very late into the project, the consortium deployed *ROLE Sandbox* (http://role-sandbox.eu) as ready-to-use sandbox for widget developers. With the original intent to analyze system uptake and quality, the maintainer integrated the MobSOS framework for community success awareness [19] including means for automated log data collection, enrichment, visual analytics, and community success modeling. Deeper data analysis soon revealed usage patterns hinting to use by other stakeholder groups such as teachers, learners, and researchers. Already the analysis of automatically collected and enriched log data provided valuable insights with respect to learning analytics far beyond the original intent. This low-effort automated approach enabled ROLE Sandbox to become the first testbed for learning analytics on widget-based PLE on a world-wide scale [20]. From a learning analytics perspective, different views on the same data would have provided different project stakeholder groups with relevant input already during project lifetime: researchers for quantifying empirical findings, developers for monitoring widget quality and uptake, maintainers for surveilling platform status, teachers for staying aware of learning indicators, etc.

Project Learning Layers

Learning Layers was an EU FP7 large-scale integrating project running from 2012 to 2016, involving 18 partners with 13 Mio Euro budget. Core work packages included R&D activities towards innovative software for scaling informal workplace learning and their evaluation. The objective of the project was to develop a set of modular and flexible technological layers for supporting workplace practices in companies that unlock peer production and scaffold networked learning. This objective was addressed with various mobile apps and social software services on top of a scalable, light-weight infrastructure that allows for swift federated deployment in highly distributed and dynamic settings. Since the start of the project, end-users from *vocational education and training (VET)* backgrounds in healthcare and construction were deeply involved in system co-design, and continue so even beyond project lifetime. This project adopted and extended several instruments previously used in ROLE. These included the DTF, a decision making body, an OSS strategy, and a tool-supported social requirements engineering process.

During the first project months, the consortium produced a *technology survey*. Unlike in ROLE, the purpose was not only to survey existing architectures, frameworks and tools supporting deployment of workplace learning solutions, but also to assess software engineering tools in use across partner institutions. Since the first-year objective was a fast, small-scale deployment of an initial infrastructure, the technology collection mainly focused on architecture solutions. From all surveyed technologies the consortium selected four architectural models and seven products for closer scrutiny, documented in an internal report.

Requirements engineering took place in parallel to the survey. In absence of dedicated tools, requirements were first elicited during face-to-face activities and from non-digital artifacts from end-user partners (e.g. storyboards), introducing the same overhead as in ROLE. The consortium thus deployed

Requirements Bazaar [17] as tool for social requirements engineering. The knowledge of workflows and needs in SME contexts in the pilot clusters was initially captured in context cards, which depicted a typical work situation and the related challenges. These context cards resulted from visits of technical project staff to the application partners during so-called *application partner days*, one of the measures taken for ensuring stakeholder engagement. For instance, a context card for a welding station might pose challenges such as, *"How can welders document their work?"* or *"How can welders receive help from others while welding?"* These challenges were (among other activities) fed into an early project-wide *design conference*. First use cases, storyboards and design ideas were generated during this event. Co-design teams involving different kinds of stakeholders (researchers, developers, end-users) emerged around design ideas and then continued to work on use cases, storyboards, wireframes, and prototypes, using agile development methods. The artifacts created and refined during these activities were mined for requirements as first activity of the project's DTF. The requirements were then transferred to Requirements Bazaar to be negotiated with typical social operations such as voting, amending and commenting. The result was a prioritized list of requirements based on votings, accessible and traceable for all project members.

Like in ROLE, the DTF was installed early and with considerable freedom in short-term development. To simplify coordination, members were holding bi-weekly online meetings to discuss current issues like new developments, peer reviews of code, bugs or integration tasks. As decision making body for cases of disagreement or long-term decisions, the consortium established an *architecture board* early in the project, consisting of representatives from all technical partners and from all design teams.

While individual institutions functioned as product owners of the software they produced, the main task of the DTF was to develop a common infrastructure and reasonable interfacing between components for seamless integration into an overall platform. The DTF also had an active role in the establishment and maintenance of a Layers *Open Developer Library (ODevL)*, including tools and best practice documentation for requirements engineering, code hosting, revision control, issue tracking, continuous integration, containerization, continuous deployment, monitoring, and analytics.

Webinars about both used and produced tools were an essential part of the ODevL. The screencasts were produced by senior DTF members to showcase important functionalities. Developers and project stakeholders were the first target audience. However, the publication of our webinar videos received on large video hosting platforms such as youTube and vimeo resulted in many views beyond project scope and thus served dissemination purposes. With the choice of GitHub as code hosting platform, the project avoided all the problems earlier experienced in ROLE. Most importantly, every component was maintained in an own repo, but under a common Learning Layers organization. Building upon earlier experience from the *licensing* discussions in ROLE, the consortium agreed on a freedom of choice among a set of compatible permissive OSS licenses like BSD or Apache. However, in later stages, commercial partners made use of their right to switch from permissive OSS to closed source, proprietary commercial licensing without further notice. Such steps effectively turned out to be detrimental to the initial goals of working open source, as awareness of constant updates from project partners and funding agencies was hampered.

As additional *stakeholder engagement activity*, the DTF organized annual hackathons [18] where internal and external developers were invited to design and code new ideas for learning tools using project relevant APIs and tools. The hackathons usually started with tutorials about specific technologies and development methods like Scrum, to get inexperienced developers on board quickly. Closed face-to-face meetings among DTF members integrated into hackathon agendas significantly helped to push forward integration tasks. During one such hackathon, the DTF collaboratively learned how to integrate their heterogeneous prototypes to use OpenID Connect as single sign-on solution. Besides having face-to-face meetings and discussing ongoing issues, the goal of these development events was to embrace external open source communities by the means of hands-on technology presentations and coding contests. Such face-to-face meetings also helped to push forward integration tasks.

As means of supporting different forms of project *evaluation*, i.e. summative vs. formative vs. developmental, as well as qualitative vs. quantitative, the project consortium pursued different learning analytics approaches. Learning from the earlier positive experience with ROLE Sandbox, the DTF designed the project's technical infrastructure in a way that all requests issued to any of the hosted REST APIs were automatically monitored and enriched by MobSOS [19]. Additional MobSOS services to collect and analyze end-user feedback and to create visual analytics dashboards over all collected data became core part of the infrastructure and were clearly announced as common good to be shared among consortium members. However, despite its early availability and several training initiatives, the monitoring and analytics subsystem of the common infrastructure was used scarcely. Retrospectively, one of the key reasons for this shortcoming was, that evaluation was not planned as explicit, dedicated task, but rather as implicit and orthogonal to the project's work package structure with several negative consequences. First, only few technical partners responsible for the backend infrastructure provided the necessary tools, data and expertise to make use and sense of them on a project-wide scale, but lacked direct contact to end-users. Second, individual application partners with direct contact to relatively few end-users rather refrained to traditional qualitative research methods and independent, yet highly redundant efforts into instrumenting their applications to produce high-level monitoring data analyzed with proprietary tools. In the end, we learned that research projects can considerably save resources by planning for a dedicated task or even work package on a common evaluation strategy, supported by a common monitoring and

TABLE I
COMMON SOFTWARE ENGINEERING CHALLENGES IN RESEARCH PROJECTS

Early Decisions	Architectural decisions with considerable scope need to be made early in the project, as it is well known that early mistakes in a project are the costliest [21]. These decisions should also include reflections on work package and task structure.
Short Cycles	In the time frame of an R&D project, development cycles must be kept short, and initial architectures and prototypes must be provided early in the project to allow for frequent refinement loops driven by research and end-user involvement.
Sustained Impact	Deployed solutions have to scale both horizontally and vertically, as project outcomes should be exploited and sustained beyond the project's funding period. Typically, exploitability and impact of project results are key performance indicators for funding agencies, although market-readiness of ICT research project outcomes is often limited [22]).
Distributed Community	The initial community is a union of individuals affiliated to different partners. It needs to act as seed for a growing and flourishing wider community involving external members in a limited amount of time.
Support Infrastructure	The development process and environment, including procedures and tools for source code management, issue tracking, software branding, and similar, need to be defined, implemented and maintained. Eventual licensing and maintenance costs and efforts may not be underestimated.
Licensing	An OSS strategy usually helps sustain and transfer project results into practice and is thus interesting for funding agencies [8]. By definition, any OSS must ship with an OSS license. In large project consortia, agreement on one license for all developed components is hard to reach. The use of third-party OSS and differing exploitation goals among partners impose the use of different, however compatible licenses.
Stakeholder Engagement	Societal software projects inherently require instruments to achieve wide coverage of stakeholder engagement. Such instruments thus need to be as inclusive and supportive as possible even for non-technical audiences.
Baseline	In research projects, the state of the art builds the baseline for all activities. "Quick & dirty" approaches ignoring existing knowledge may thwart the cutting-edge research ambitions of the project. However, software engineering processes need to be flexible to understand and adapt to emergent change, resulting from the complex and never ending interplay of people influencing technology and vice versa, as we find it in socio-technical systems and as it is described by ICT-related adaptations of structuration theory [23].
Unknown Territory	Research outcomes inherently remain unspecified beforehand, while the research methodology leading to those expected outcomes must be well-defined. This poses a concrete challenge, as software engineering activities need to synchronize with primary research activities.
Success Awareness	Any software artifact developed within research projects requires rigorous evaluation with respect to success as complex construct combining quality and impact as they are perceived by different relevant stakeholder groups. With respect to sustainability and given that success itself is a highly dynamic and context-dependent construct, success awareness must be conceptualized as result of ongoing long-term developmental evaluation even beyond project lifetime [9], [19].
Decision Making	Although research institutions collaborate in projects with legally binding contracts and agreed work plans, all of them have different motivations, internal agendas and working processes to be aligned with the overall project agenda. This diversity often poses an obstacle to efficient decision making.

analytics infrastructure trusted by all evaluation partners. Such a dedicated task must allocate sufficient own resources, such that partners can reach agreement and achieve an appropriate adaptation of the analytics infrastructure to the particular evaluation interests and needs of individual partners as well as the project as a whole.

As result of ongoing reflection in ROLE, Layers, and several smaller research projects, we identified a set of challenges common in societal software engineering-related research projects (cf. Table I). In the following, this collection of common challenges serves as basis for developing a sustainable societal software engineering methodology and infrastructure suitable for research projects.

IV. DEVOPSUSE FOR RESEARCH PROJECTS

Meeting the challenges extracted in the previous section requires a sound methodological basis and technical support infrastructure to sustainably design and guide software engineering processes within and beyond the boundaries of research projects. The approach we present here builds on three fundamental observations. First, in academic settings, both development and operations are carried out by the same researchers, in contrast to distinct departments in company contexts. Second, despite its obvious importance for tasks such as requirements engineering, testing, and evaluation, end-user participation remains unreflected or implicit, while its importance is gaining wider attention in both industrial and academic research settings. Thus, end-user participation from different community contexts must be made more explicit in societal software engineering to gain best possible coverage and awareness for the plethora of diverse stakeholder groups involved in societal processes. Third, the wider OSS community has continuously evolved software engineering best practice, well-known and accepted among professional developers across projects and problem domains. Furthermore, providers of professional software engineering tools are supportive to the Open Source philosophy by granting free licensing to OSS projects. As such, OSS development is inherently well-suited for application in research projects, in particular with respect to continued development after project end, either in follow-up projects, dedicated OSS communities, or commercial exploitation.

The left part of Figure 1 shows our DevOpsUse methodology as an extension of the abstract DevOps [24] life cycle by an additional *end-user cycle*. The innermost circle reflects the standard DevOps life cycle as baseline. DevOps essentially postulates a vivid collaboration culture among software *development (DEV)* and *operations (OPS)*, supported by a highly integrated and automated tool infrastructure. We use culture here as a synonym for practices and not in the sense of Hofstede [25].

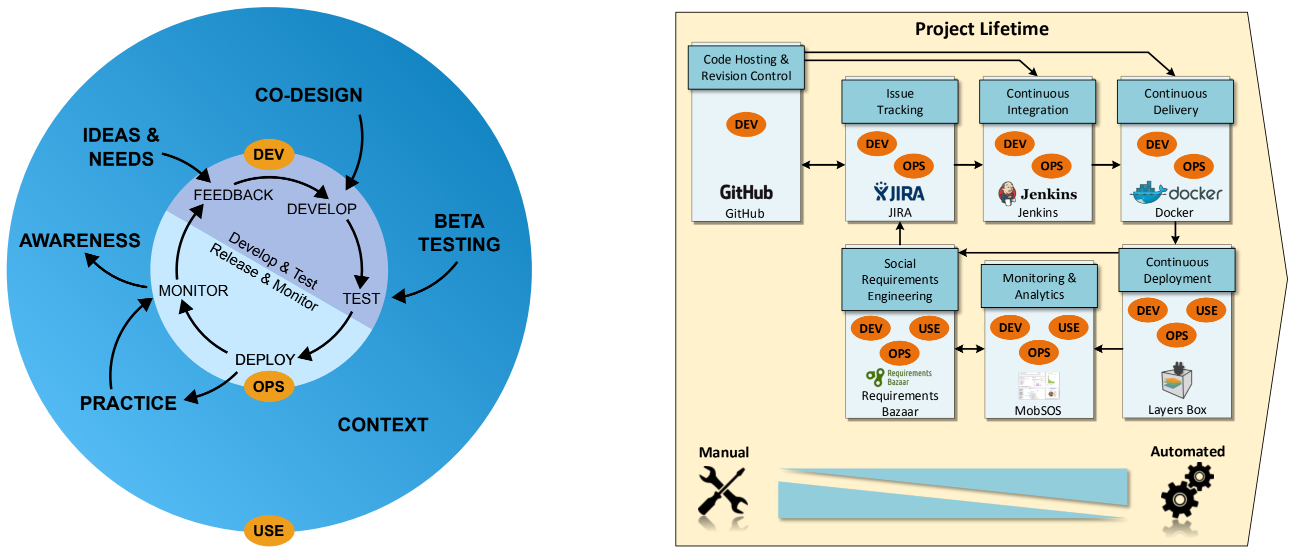

Fig. 1. DevOpsUse life cycle (left) and tool support instrumentation roadmap (right)

In order to appropriately reflect the importance of end-user contributions to societal software engineering, we add a USE ring in parallel, illustrating end-user activity throughout different phases of the DevOps cycle. Particularly in early project phases, potential end-users of societal software are invaluable resources for generating and negotiating innovative ideas and determining relevant requirements. However, often the access to end-users is hard to achieve for project developers. In the same vein, getting end-users to participate in co-designing and alpha/beta testing long before any official release is desirable, but challenging. Deployment is partially guided by end-users' requirements with respect to their contexts. In particular, end-users help decide in which premises societal software is deployed (i.e. public cloud data centers, organizational cloud installations, private clouds, or hybrid forms [26]). Once deployed, even in early and immaterial prototype phases, end-users carry out their practice by using the software. With their use, end-users generate traces feedback on quality and impact that can be monitored, analyzed, visualized in aggregated forms suitable to create awareness [19] for all stakeholders, again including end-users. The cycle closes with the repeated negotiation of requirements as part of social requirements engineering [17] to refine existing features or find new ideas.

Technical DevOpsUse Instruments

Inline with standard DevOps methodologies, our DevOpsUse methodology is accompanied by an integrated infrastructure of supportive services and tools from best practice in OSS development. In addition to the DevOps goal of achieving the highest level of integration and automation for such an infrastructure, our DevOpsUse infrastructure emphasizes the participation and awareness of end-user communities across projects. The result is a combination of state-of-the-art technical DevOps instruments commonly used in OSS development projects, augmented with a few, yet highly influential instruments dedicated to the inclusion of end-user communities without strong technical background. In the right part of Figure 1, we provide an overview of the proposed DevOpsUse tool set. Following the observation that DevOpsUse infrastructures are built incrementally, we provide this overview in a roadmap fashion. For each tool, we highlight which groups are mostly involved and profit (Dev/Ops/Use) and provide a best practice choice. With each tool added to the infrastructure, we achieve another leap towards automation. Most importantly, tools in the same class are often not interchangeable, but introduce compatibility dependencies expressed by arrows. These dependencies can quickly narrow down the choice for downstream tools. The first and most important decision is the code hosting and revision control system for developers. Most other tools downstream in the roadmap depend on this choice. We made best experiences with git-based systems such as GitHub or its OSS pendant GitLab, being de-facto standard among OSS communities. Likewise, we recommend JIRA for issue tracking, Jenkins for continuous integration and Docker for containerization. Finalizing our roadmap, we propose boxed solutions for the continuous deployment of applications that end-user communities can choose from app stores. In the next paragraphs, we focus on two DevOpsUse instruments particularly designed for end-user involvement, i.e. *social requirements engineering* and *monitoring & analytics*.

Social Requirements Engineering. Traditional requirements engineering techniques such as focus groups require co-presence of researchers, engineers, and end-users and efforts in traceably post-processing results. Across projects, we found such practices unsustainable in terms of excessive traveling and personnel costs. We furthermore found that end-users conceive issue trackers as intimidating due to their technical complexity and jargon. We therefore developed a large-scale *social requirements engineering (SRE)* approach, supported by Requirements Bazaar, an open source social software platform for requirements negotiation among non-technical end-user audiences and professional developers. In order to guarantee traceable awareness on the development process, we realized a two-way integration involving automated data and communication flow between Requirements Bazaar (cf. [17]) and an issue tracker. Once a developer commits to the realization of a requirement in Requirements Bazaar, the requirement including all its attached comments and artifacts is pushed into the issue tracker. Subsequent updates on the development process in the issue tracker relevant for end-users are pushed back to Requirements Bazaar to keep end-users aware of development progress.

Monitoring & Analytics. In order to create the basis for short-term or even longer term awareness on quality and impact, contemporary deployment infrastructure is instrumented with various means for and forms of monitoring and analytics. For example, monitoring frameworks are installed for the automated collection and contextual enrichment of usage data, while social media channels serve as data sources for human-generated feedback. Different data sources are triangulated, filtered, processed and visualized in analytics dashboards to ultimately create awareness for wide arrays of stakeholder-dependent target metrics, e.g. infrastructure sanity for operations or user counts for dissemination. State-of-the-art analytics frameworks are usually of myopic focus limited to standard metrics for development, operations, and management. However, the analysis of complex phenomena emerging from the use of the designed systems is an essential task in research projects, with competing notions on relevant quality and impact factors within and across end-user communities. We address this gap with our MobSOS framework for achieving shared community success awareness by negotiating individual stakeholder notions of success with the help of fluid success models [19].

Social DevOpsUse Instruments

In addition to technical DevOpsUse instruments, our methodology includes a set of seeding social instruments, effectively preparing research projects for seamless transition to independent OSS communities after project end. First, these social instruments must enforce setup, maintenance and continuous adaptation of the aforementioned infrastructure for sustained successful use. Second, these instruments must actively pursue dissemination and exploitation strategies to secure resources for sustained long-term development and operations. They must reach out to multipliers influential in mobilizing end-user communities and external developers for sustained development and operations. These social instruments are subject to the organization of virtual teams, ranging from decentralized self-coordination to centralized venture teams [27]. Taking into account the motivational situation and the management structures at research centers and universities typically involved in collaborative research projects, blended approaches are recommended. According to our findings, a combination of decentralized short- and mid-term self-coordination, with a system architecture core team installed to deal with long-term, high-impact decisions and strategic development objectives works best in large research projects. For the decentralized self-coordination we made best experience with a distributed team structure, the *Developer Task Force (DTF)*. For the core architecture control we made best experience with a governing body, the *Architecture Board*.

Developer Taskforce.

Splitting the software development effort in a research project into artificial teams should be avoided, as part-time development in small distributed teams will hamper the progress [28]. To facilitate the emergence of a team spirit with shared ownership, we recommend to build one single virtual team early into a research project. In line with previous experience (cf. Section III), we recommend a *Developer Task Force* as informal community to bundle developer resources from distributed partners in a virtual team structure. Typically, projects involve either professional developers from participating companies or researchers/PhD students at academic institutions. In either case, developers usually do not work full time on a single project. A unified, virtual task force shall help to sustain a steady heartbeat [14] in the face of these challenges. Task force members are usually key contributors in the project software engineering life cycle. They are involved in decision making for choosing the right technologies to solve architectural and implementation problems. They are also the closest stakeholders that can ensure traceability and a good balance between project requirements, decisions and implementation status. Moreover, task force members bring in their professional experience to influence decisions on employed software engineering methods and related support infrastructure. The task force can be seen as an exploration unit with clear competences, which can take initiative in proposing technical solutions to project managers. Even though the developer task force is ultimately subject to internal project policies and reports to the leading structures, the participation and influence of senior project members is discouraged to facilitate informal knowledge exchange among junior researcher and developer peers, and to establish a space for experimenting with prototypes and innovative solutions in the scope of the project's development agenda. The task force should be self-managed and given considerable autonomy for defining, adopting and achieving short- and mid-term objectives and practices. This will lower the pressure coming from rules and regulations, known to be a failure factor [14].

To obtain an overview of relevant existing technologies, an established initial activity of the developer task force is to conduct an initial *technology survey*. Existing technology options are surveyed by the technical partners and documented internally. Goal is to explore, assess, negotiate, and ideally reach early consensus among the distributed teams involved in the task force. In early cycles, this activity will help to remedy challenges related to architectural decisions. Also, such a survey will help to reveal whether licensing models of adopted components and software are compatible with the project's licensing model.

Architecture Board. Projects should further establish a governing body as authority to make binding decisions for all partners. We propose an *Architecture Board* to reflect its duty of making and enforcing global decisions on technical architecture of software artifacts and engineering processes. This is particularly useful when development process and conventions are set up upfront [28]. The authority should enforce policies and be the only entity allowed to change policies. Moreover, we suggest to establish this authority in the project contract—either as a dedicated governing body or assigned to an existing board, e.g. the project management board—to make it a legally binding instrument for making project wide decisions on all software engineering related issues. The Architecture Board will also help to increase stakeholder commitment due to stronger involvement in decision making. Informed, consensus-oriented, and forward-directed decision making, optimally supported by consulting instruments, increases the chance of achieving sustained post-project exploitation and impact.

External Involvement.

Large research projects usually face a challenge in meeting the goals of multiple stakeholder groups. Researchers aspire high-quality publications, developers target an efficient development process, and application partners wish for readily employable software. Projects must therefore provide platforms to engage stakeholders and allow for communication. We made best experiences with establishing *co-design teams*, consisting of representatives from end-users, application partners, researchers, and developers. These permanent, open work groups elaborate design ideas aligning software prototypes with requirements stated by end-users or their application partner proxies. Main concern of these design teams is to drive technological innovation based on real problems and scenarios. They produce usage scenarios and wire frames significantly helping to reduce development efforts, as they directly formulate concrete demands to prototypes. Including end-users by welcoming ideas and requirements additionally raises stakeholder engagement and more informed decision making, as it explicitly considers particular end-user impacts, and not only impacts beneficial for the research project and its partners. With a wide variety of different stakeholder notions, finding commonalities and pivotal elements across these ideas is key to success. A constant dialogue between design teams and developers is maintained by assigning at least one design team representative to the DTF. Thereby, current endeavors of the co-design teams can be directly transferred to the DTF meetings and vice versa.

Especially with respect to sustainability beyond project life time, projects should start initiatives towards building a vibrant developer community around project offerings. Again, we made best experience with providing the DTF with freedom to take active responsibility in pushing sustainable community building in different kinds of public appearances, especially at research conferences and OSS meetings. Such activities should even start at early stages into development, despite an often found reluctance of consortium members to present preliminary, often intangible outcomes. The common sense mantra *"Build it, and they will come"*, often purported by developers is the wrong approach. Instead, project members should follow the mantra *"Show me what you got"*, as it is especially important in the OSS community to gain trust and credibility.

Demo sessions, hands-on workshops, or competitions at conferences usually attract younger, development-oriented researchers from different project contexts and thus facilitate active exchange and cross-fertilization. OSS developer meetings usually offer a wide range of formats, e.g. exhibition booths, lightning talks, or theme rooms. The motivation to appear at such events is not only to disseminate project results, but mainly to learn from rapidly evolving best practice in the OSS scene, and to search for high-profile OSS projects bearing potential to sustain project results under their umbrella.

Last, but not least, we made best experience with hosting own community building events, e.g. in the form of on-site hackathons or remote development contests. Developer task force members usually provide for a convenient and welcoming development environment and serve as assistance for external developers, introducing them into respective offerings of the research project and inviting their contribution.

V. Lessons Learned

The experiences about the software engineering process reported here are the synopsis of years of work in creating and sustaining results within and beyond the scope of a funded research project. As part of the European research community, we see this as an opportunity to facilitate software engineering in future research grant programs, e.g. in the EU Horizon 2020 programme, which seem to be even larger and more product-oriented than the integrated projects of the EU FP7 programme. Here, we present the key lessons we learned.

People. The software engineering process in research projects is largely a social one, involving many stakeholders with different goals and agendas. Even with a good project plan, requirements and priorities of stakeholders are subject to emergent change over time [23]. These changes must be captured and traced as good as possible. Since most of the knowledge of the people is opaque in the beginning, there must be sufficient opportunity to meet and give voice to the stakeholders. Often it is argued that the innovative ideas are coming from the researchers in the project. This may be true to some extent, but ideas are forlorn if stakeholders are not willing to adopt them, usually described as *"not-invented-here"* syndrome. Traceability of social processes is even more important in the light of Open Source Software communities being subject to generation changes within their life cycles [29].

Open Source Development. It is essential to make a strong commitment towards open source development. Many research projects are understaffed with experienced developers. Thus, outreaching to the OSS communities should be integral part of project dissemination from the very beginning. However, it is an illusion to think that a one-shot approach is working. Experienced and established OSS communities have subtle means for checking the desired long-time commitment of developers. In the same manner, research projects pursuing OSS strategies should use such means to identify and nurture commitment of external parties. As a consequence, also project developers need training in OSS development and sufficient freedom to participate in an OSS community and eventually become a valuable member. An issue, which is usually ignored in the beginning, is the OSS licensing models. These turn out to bring up complicated administrative and legal routines in all involved companies and institutions.

Automation & Integration. Project partners should be able to focus on true communication and collaboration. Therefore, partners should make best efforts to establish tool chains minimizing the need for manual intervention in highly repetitive tasks, e.g. regression testing and automated builds triggered by new events from revision control, as well manual transfer of information from one system to the other, e.g. copying commit messages belonging to an issue from revision control to an issue tracker. Most providers of DevOps tools have recognized this key requirement for automation and integration and clearly document their compatibility with other tools. However, partners must analyze particular compatibilities during upfront technology surveys to avoid later integration issues and automation gaps, effectively resulting in unnecessary manual work again. Interestingly, even tasks like compliance checking become automizable with new standard formats for communicating components, licenses, and copyrights associated with software packages, e.g. *Software Package Data Exchange (SPDX)*, thus avoiding licensing problems early.

Awareness. Creating and sustaining awareness of software engineering activities as well as the evolution of software artifacts as part of project research work is of high importance. Creating awareness involves events where project members are showing up or which are organized by the project for branding the project (for instance, competitions or developer camps). Awareness also means to provide tools for reflection in the developer community. In particular, tools for monitoring and analytics are key prerequisite to awareness, but still require resources allocated to training and the active reflection of needs, interests, and outcomes between analysts and stakeholders. Awareness means moreover a process of becoming for the members of the development community.

Time. Time is an essential factor. It is paramount to start very early in shaping and providing the development infrastructure and grow continuously. In that sense, a pre-configured development infrastructure like the one drawn in our DevOpsUse methodology, which can be rapidly deployed for a new project, is a better choice than starting from scratch. However, research project consortia should not underestimate the lack of capacity in some project members spotting issues in a common project infrastructure. Such issues should be openly discussed and resolved, if necessary with the help of an architecture board. Unresolved issues of that kind often lead to separated installations of alternative infrastructures at individual partner institutions and in consequence to integration issues that are often discovered very late into the project. For every partner there is a certain entry barrier to new tools and infrastructures, either of financial nature due to licensing costs or of social nature due to lack of training, experience, or interest. Consequently, we built our infrastructure mostly on widely used and well-established open-source, industry strength products.

VI. Conclusion

In this paper, we discussed the essential research gap of lacking methodologies and technical infrastructures for sustainability-aware software engineering in collaborative research projects. We first discussed two longitudinal case studies in large-scale EU research projects and extracted common challenges. Based on our empirical findings, we then presented our DevOpsUse methodology as an agile software engineering methodology building upon DevOps, OSS development practice, and with a strong focus on end-user inclusion and awareness. We furthermore presented our DevOpsUse infrastructure as a highly integrated and automated set of freely available OSS support tools, as well as social instruments for sustainable community building. We finally contributed a set of lessons learned with the intention to inform future projects considering the uptake of a DevOpsUse methodology.

However, our results bear several limitations. First and most obvious, our case studies were based on two large scale and several smaller scale projects we were involved in. Although conversations with representatives of other projects exhibited very similar challenges and patterns of evolution with very different partner constellations, we see the clear need for more case studies to replicate and further generalize our findings. Second, not all challenges and instruments of our DevOpsUse methodology apply for all projects. Especially in smaller projects, resources do not suffice to establish the complete methodology and infrastructure. We are currently observing how partial configurations of our DevOpsUse methodology apply in smaller research projects. Third, our methodology only applies to research projects compliant with an open research philosophy. As such, confidential research projects and/or research projects producing closed source implementations inherently disqualify for the application of the full DevOpsUse methodology, while DevOps remains possible. Finally, as part of the everlasting duality between social and technical development [23], we must acknowledge, that any software engineering methodology will evolve in parallel with the software artifacts it produces and all involved people, thus gaining new insights from project to project in a double-loop learning fashion. As such, we continue our endeavor to apply longer term community data collection and analytics in order to gain a deeper understanding of the current and future evolution of sustainable software engineering practice in research projects across communities, including the most relevant universally valid quality and impact factors [19].

With this contribution we aim to preserve previously and currently successful practice in a way that serves future project consortia as shortcut in planning, establishing, and maintaining their software engineering processes, either as a dedicated work package or by picking a subset of the instruments and activities that are tailored to their needs. Many research projects seem to currently reinvent the development wheel independent of each other, thus wasting precious resources, particularly during the early forming stages of a project's processes. With this work we eventually want to establish a culture of sharing and continued refinement of sustainable software engineering best practices in and across research projects and communities focusing on societal challenges.

Acknowledgments

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme projects "Responsive Open Learning Environments" (grant no. 231396) and "Learning Layers" (grant no. 318209), as well as the European Union's Horizon 2020 Programme through the project "WEKIT" (grant no. 687669). We thank all project partners for collaboratively shaping the DevOpsUse methodology.

References

[1] European Commission, "What is FP7? The Basics," 2015. [Online]. Available: https://ec.europa.eu/research/fp7/understanding/fp7inbrief/what-is_en.html (last access: Feb 2017)

[2] United Nations and Accenture, "The United Nations Global Compact-Accenture Strategy CEO Study 2016: Agenda 2030: A Window of Opportunity," UN Global Compact, Tech. Rep., 2016.

[3] R. Chitchyan, C. Becker, S. Betz, L. Duboc, B. Penzenstadler, N. Seyff, and C. C. Venters, "Sustainability Design in Requirements Engineering: State of Practice," in *Proceedings of the 38th International Conference on Software Engineering Companion*. New York, NY, USA: ACM, 2016, pp. 533–542.

[4] European Commission, "Budget FP7 Research Europe," 2013. [Online]. Available: http://ec.europa.eu/research/fp7/index_en.cfm?pg=budget (last access: Feb 2017)

[5] National Science Foundation, "FY 2014 NSF Budget Request to Congress," 2014. [Online]. Available: http://www.nsf.gov/about/budget/fy2014/pdf/18_fy2014.pdf (last access: Feb 2017)

[6] European Commission, "Open Source Strategy in the European Commission," 2016. [Online]. Available: http://ec.europa.eu/dgs/informatics/oss_tech (last access: Feb 2017)

[7] T. Scott and A. E. Rung, "Federal Source Code Policy - M-16-21 Memorandum for the Heads of Departments and Agencies," 2016. [Online]. Available: https://sourcecode.cio.gov/OSS/ (last access: Feb 2017)

[8] European Commission, "Free and open source software activities in European Information Society initiatives," 2015. [Online]. Available: http://cordis.europa.eu/fp7/ict/ssai/foss-home_en.html (last access: Feb 2017)

[9] C. Becker, R. Chitchyan, L. Duboc, S. Easterbrook, B. Penzenstadler, N. Seyff, and C. C. Venters, "Sustainability Design and Software: The Karlskrona Manifesto," in *Proceedings of the 37th International Conference on Software Engineering - Volume 2*. Piscataway, NJ, USA: IEEE Press, 2015, pp. 467–476.

[10] 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.

[11] J. C. Carver and T. Epperly, "Software Engineering for Computational Science and Engineering," *Computing in Science & Engineering*, vol. 16, no. 3, pp. 6–9, 2014.

[12] J. C. Carver, R. P. Kendall, S. E. Squires, and D. E. Post, "Software Development Environments for Scientific and Engineering Software: A Series of Case Studies," in *Proceedings of the 29th International Conference on Software Engineering*. IEEE Computer Society, 2007, pp. 550–559.

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

[14] H. Huijgens, R. van Solingen, and A. van Deursen, "How to build a good practice software project portfolio?" in *36th International Conference on Software Engineering, ICSE '14, Companion Proceedings*, P. Jalote, L. Briand, and A. van der Hoek, Eds. ACM, 2014, pp. 64–73.

[15] N. A. Ebrahim, S. Ahmed, and Z. Taha, "Establishing Virtual R&D Teams: Obliged Policy. CoRR, abs/1208.0994," 2012. [Online]. Available: http://arxiv.org/abs/1208.0944 (last access: Feb 2017)

[16] 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.

[17] D. Renzel, M. Behrendt, R. Klamma, and M. Jarke, "Requirements Bazaar: Social Requirements Engineering for Community-Driven Innovation," in *2013 21st IEEE International Requirements Engineering Conference (RE) Proceedings*. Los Alamitos, CA, USA: IEEE Computer Society, 2013, pp. 326–327.

[18] M. Komssi, D. Pichlis, M. Raatikainen, K. Kindström, and J. Järvinen, "What are Hackathons for?" *IEEE Software*, vol. 32, no. 5, pp. 60–67, 2015.

[19] D. Renzel, "Information Systems Success Awareness for Professional Long Tail Communities of Practice," Doctoral Dissertation, RWTH Aachen University, Aachen, Germany, July 2016. [Online]. Available: http://publications.rwth-aachen.de/record/667644/files/667644.pdf (last access: Feb 2017)

[20] D. Renzel and R. Klamma, "From Micro to Macro: Analyzing Activity in the ROLE Sandbox," in *Proceedings of the Third International Conference on Learning Analytics and Knowledge*, D. Suthers, K. Verbert, E. Duval, and X. Ochoa, Eds. Leuven, Belgium: ACM, 2013, pp. 250–254.

[21] 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.

[22] European Commission, "FP6 IST Impact Analysis Study: Final Report," 2009. [Online]. Available: http://cordis.europa.eu/fp7/ict/impact/documents/wing-pilot-fp6-final-report-18-12-09.pdf (last access: Feb 2017)

[23] W. J. Orlikowski, "The sociomateriality of organisational life: considering technology in management research," *Journal of Economics*, vol. 34, pp. 125–141, 2010.

[24] J. Davis and K. Daniels, *Effective DevOps - Building a Culture of Collaboration, Affinity, and Tooling at Scale*. Sebastopol, CA, USA: O'Reilly Media Inc., 2016.

[25] G. Hofstede, G. J. Hofstede, and M. Minkov, *Cultures and Organizations: Software of the Mind*, 3rd ed. McGraw-Hill USA, 2010.

[26] P. Mell and T. Grance, "The NIST Definition of Cloud Computing - Recommendation of the National Institute of Standardization and Technology," NIST, Tech. Rep., 2011.

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

[28] P. Kunszt, "Grid Middleware Development in Large International Projects - Experience and Recommendations," in *International Conference on Software Engineering Advances (ICSEA 2007)*. IEEE, 2007, pp. 82–86.

[29] K. Neulinger, A. Hannemann, R. Klamma, and M. Jarke, "A Longitudinal Study of Community-Oriented Open Source Software Development," in *Advanced Information Systems Engineering: 28th International Conference, CAiSE 2016, Ljubljana, Slovenia, June 13-17, 2016. Proceedings*, S. Nurcan, P. Soffer, M. Bajec, and J. Eder, Eds. Cham, Switzerland: Springer International Publishing, 2016, pp. 509–523.