Together, Yet Apart
The Research Prototype Architecture Dilemma

Falko Koetter, Monika Kochanowski, Florian Maier and Thomas Renner
Fraunhofer IAO, Stuttgart, Germany

Keywords: Software Engineering, Service-oriented Architecture, Service Platforms, Distributed Systems.

Abstract: Distributed research projects combine the know-how of industry and research partners from different organizations and countries. In IT, joint software development of research prototypes is an integral part of such projects. However, project partners have individual interests in the developments, ranging from creating new projects to finishing a PhD thesis. This leads to a dilemma - components need to work together within the projects, but have an individual purpose apart from the projects. In this work, we investigate the impact of research project characteristics, in particular the aforementioned dilemma, on software architecture in research projects. From expert interviews, we identify unique architectural challenges inherent to research projects. In this position paper, we argue that these challenges must be considered when planning and executing research projects.

1 INTRODUCTION

Every year, the European Union and national governments grant millions of Euros for research and development in IT. All over the world research projects typically are joint ventures, with consortiums of industry and research partners. These consortiums come together in the project, researching, developing software prototypes and evaluating them. Work on the prototype is distributed between partners according to capabilities and interests (Winkler et al., 2013). After the project, the consortium may dissolve and keep their parts of the prototype for further use. This leads to a dilemma: For a successful research project, a well-integrated and functional prototype needs to be developed.

However to keep components usable after the project, they need to be *independent* and fit for their purpose after the project. Thus, organizational and political considerations become a part of software architecture, leading to challenges in research prototype development.

In this work we will describe these challenges in detail based on literature and interviews. For this research, we conducted expert interviews with 10 researchers and software developers with experience in software engineering from 8 different distributed research projects.

The remainder of this work is structured as follows. Section 2 gives an overview of software engineering and architecture in distributed projects in general and research projects in particular. Section 3 gives a motivational example of a research prototype. Section 4 describes the characteristics of distributed research conflicts including premises, goals and processes. In Section 5, the impact of these characteristics on software architecure and the resulting issues are described. Finally, Section 6 gives a conclusion and an outlook on future work.

2 RELATED WORK

In this section, we will investigate related work on distributed software development and existing work on software development in research projects.

(Grinter et al., 1999) investigates the impact of geographically distributed software development using expert interviews. Different kinds of project and organization structures are investigated. It is shown that clear assignment of responsibility is a critical success factor. Negative impacts include the lack of informal communication, the making of decisions by central entities without investigating the impact for other stakeholders and higher efforts for integration and error fixing. It is also shown that ambiguity of requirements, responsibilities, etc. is a problem as it cannot be resolved easily between geographically distributed

Koetter, F., Kochanowski, M., Maier, F. and Renner, T.
Together, Yet Apart - The Research Prototype Architecture Dilemma.
DOI: 10.5220/0006370106460653
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 618-625
ISBN: 978-989-758-243-1

teams.

(Lawrence, 2006) presents a case study of a distributed research project between 9 US universities. Findings include the conflict between research goal and software development and the conflict between the need for a reliable product versus the need for an innovative proof-of-concept. Research projects lack clear authority structures and a central authority. If project managers are not seen as an authority their decisions may not be followed. The case study stresses the importance of a common goal. While some of these findings are applicable to the topics of this work, a key difference is that no industry partners participated in the project and a common software product was developed.

In (Spencer et al., 2011) an overview of challenges and opportunities for the management of distributed research projects is given. The research shows the importance of team building and a common goal, which may be issues due to the ad-hoc nature of consortiums in the grant process. Also, the lack of a central authority is an issue.

(Howison and Herbsleb, 2013) investigates incentives in scientific software development, showing that while reputation and academic credit may be sufficient motivation for software development; other incentives (e.g. financial) are needed for researchers to integrate prototypes.

Methods for collaborative software engineering in distributed research projects are outlined in (Derntl et al., 2015). Identified challenges include different interests of stakeholders, decision making and stakeholder commitment. Suggested measures aim to lead to a consensus and a convergence of software engineering efforts. In regards to software architecture, an *Architecture Board* with representatives of all stakeholders is suggested, making binding architecture decisions. While the suggested measures present well-proven practices, it is questionable if they can be enforced in all research projects in spite of different existing practices and limited resources. Informal communication is critical but difficult across sites.

(Winkler et al., 2013) investigates the differences between research prototypes and products. Identified issues include conflicts of interest, unclear and changing requirements and unstable software architecture. As a research prototype typically is limited in regards to stability, robustness, usability and fault tolerance, these qualities need to be added in a quality-oriented second development phase.

(Nguyen-Duc et al., 2015) offers a comprehensive literature review of globally distributed software development, identifying common issues like less frequent communication, large number of communication partners, difficulty in scheduling meetings, the aversion to change processes to align with partners, difficulty in sharing responsibility across different partners and the importance of coordination across company boundaries. However, this literature review did not find work related to research projects in particular.

An older literature review regarding virtual teams, i.e. geographically and/or organizationally distributed team teams, is given in (Powell et al., 2004). Identified issues include the importance of informal communication and knowledge sharing, the possible lack of social relationships between team members, the importance of trust and common goals. Results comparing the performance and satisfaction of traditional and virtual teams are mixed.

The issues and possible solutions identified in related work versus their applicability in research projects are outlined in Table 1. Two things can be shown: most of the issues on distributed development of software are also issues in research projects, but not all. Many typical countermeasures are difficult to enforce in research projects due to the distributed organization and budget constraints. In the following we will outline the unique premises and challenges in research projects.

3 MOTIVATIONAL EXAMPLE

A motivational example derived from previous research projects is shown in figure 1. It shows an application scenario from the internet of things, where certain physical items, so-called tools, can be booked, dynamically scheduled, and secured for access management. Software vendor 1 (SV1) has an existing product for the management of these tools. Within the project scope, this is used as a basis for extension with online booking components. A University (UNI) is performing research on dynamic allocation of things in the internet of things, which has to integrate with the booking subsystem. Another software vendor (SV2) is specialized in security and focuses on related tasks. Using RFID technology, SV2 develops software and some minor hardware extensions for access management for the tools. Worker data for end users (USER) already exists in a payroll system, which has to be integrated into the developed software. Additionally workers have to be notified when changes in the booking of their tools arise, but it is unclear who will implement this component.

Table 1: Issues and possible solutions versus their applicability in research software development projects.

Typical problem	Typical solutions	Comments on applicability in research projects	Applicability
Distributed software development	Clear assignment of responsibility	No central authority	Difficult
Lack of informal communication	Common toolset	Decision on toolset based on different cultures and budget constraints	Moderate
Lack of informal communication	Geographic proximity	Importance of project meetings for doing *actual software development*	Moderate
Lack of informal communication	Teambuilding measures	Takes long time with many partners (depending on size)	Applicable
Lack of informal communication	Process (agile development methods)	Decide on one development process (depending on experience)	Moderate
Decisions by central entities without investigating the impact	Better decisions	No central entity	Not a problem
Ambiguity of requirements	Good specification	Difficult in research context due to nature of research	Difficult
Lack of common goal and different interests	Ensure common goal in project definition	Evaluation of projects drafts does not focus on this point	Difficult
Lack of incentives for prototype integration	Financial, academic, etc.	Should be applied to show impact on results	Difficult
Stakeholder commitment	Architecture board to force consensus	Architecture as a compromise, does not ensure best possibility	Moderate

4 CHARACTERISTICS OF SOFTWARE DEVELOPMENT IN RESEARCH PROJECTS

Compared to software product development, i.e. the development of a software product for internal or external users, software research prototype development operates on different premises, processes and goals. Each of these aspects will be analyzed in the following subsections.

4.1 Premises

Whereas in software product development software is developed for one or multiple customers from which *requirements* stem, in a research project no clear responsibility for requirements is defined. While research grants are given on the basis of a grant proposal, thesis proposals are not as detailed as a requirements specification, not least because research by its very nature is not as predictable as product development. Therefore, other sources of requirements are used. Requirements can be gathered from pilot users,

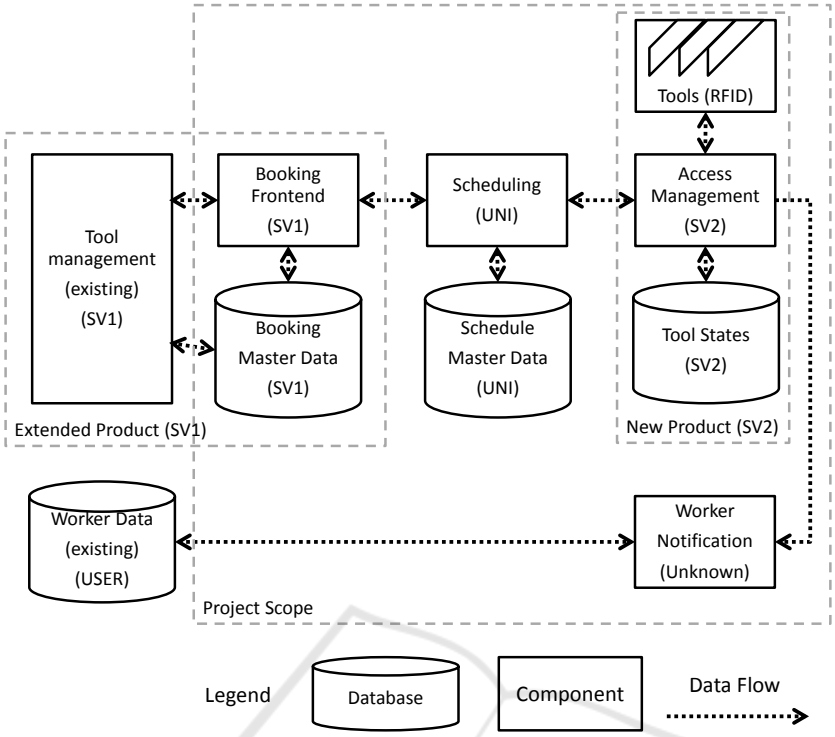

Figure 1: Architecture of example tool management research prototype.

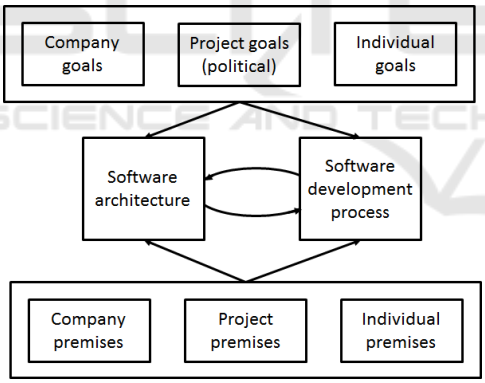

Figure 2: Goals, premises, architecture and process in research projects.

surveys, literature research, the research questions to be answered and the intended future use of the prototype, which varies between the different stakeholders. Compared to conventional customers, the funding bodies are no source for requirements, as they are more concerned about the impact of the project rather than personal use of the results.

As there are multiple sources of requirements, there is also no central *product owner* for the research prototype. Mostly this is resolved by the project manager. However, the project manager will not use the resulting software and is from one of the involved companies, therefore has to balance the needs of his company team, the complete project team, the project goal, company goals, etc. As in research projects each company is mainly responsible for its own results, differences in opinion need to be negotiated between project partners under the lead of the project manager. Typical examples include the responsibility for implementation of shared features and the implementation of additional features not covered by the initial grant proposal or features which are needed in the project software, but not specified within one partner.

As software development teams consist of employees from all project partners, they are geographically and organizationally distributed. Depending on the partner type, the software development *experience* of the teams may differ. Researchers do not always see themselves as software developers, whereas software companies do. In contrast to product development projects, where it can be largely assumed that nearly all involved developers want to improve in the field of software development and see it as their main responsibility to provide good quality code, this might differ in research projects (see also subsection goals).

Additional premises include the software development process companies used before, tools that should be used, individual skill levels, project premises like milestones defined in the project, etc.

The tools and process developed have to be negotiated - but as each company largely works on its own, usually a two-step development process evolves: (1) development of the interfaces which concern more than one partner (2) development of the features that concern only one partner. For (1), one solution might be a workshop which defines the interfaces and the implementation plan (waterfall model), whereas for (2) one company might use Scrum, and another one no process at all because only one person is involved.

4.2 Goals

While the goal of software product development is software, in a research project the resulting software is one of multiple goals.

For the funding bodies, goals include the quality of research, the scientific impact, the economic potential as well as the positive publicity. Typically, results and progress are documented in reports. These reports may contain descriptions and results of developed software, but the software itself is not delivered.

For the project partners, the importance of software may vary. For industry partners, the software may be the cornerstone of potential future products or may be integrated into existing products. For research partners, software may just be a means to answer the research questions without any planned reuse after the projects. Often, individual researchers' foremost goal is to complete research papers or a dissertation, deemphasizing software quality. Thus, the lifecycle of software varies between different project partners. For some, the prototype is a long-term *result* of the research project while for others the prototype is only a *means* to achieve results. Therefore, expectations of software quality, especially regarding usability, resilience and tolerable bugs will differ.

The experts noted that the different goals of the project partners and their project team members are one important source of architectural and software development process opinions, negotiations, and decisions.

4.3 Processes

The typical software development process consists of several, more or less similar steps: Strategy, Analysis, Concept, Implementation, Test, Go-Live (Jacobson et al., 1999). In agile development these steps are repeated to iteratively develop a prototype, emphasizing steps like implementation and test while spending less time on specifications, etc. (Beck et al., 2001). However, research project goals include usually some kind of textual documents (so-called deliverables), where effort has to be spent on specifications. Based on these, the development starts, but - as already known in traditional software development models - these documents are produced once typically and not adapted anymore - the software outdates the documents at some point in time.

Development processes in research projects of the interviewees typically followed a loosely coupled iterative process. Software components are developed by partners and then integrated, e.g. at the end of work packages or for important milestones like model trials. While these processes are iterative, they are not completely agile. One factor is the lack of coordination and transparency in planning tasks of a partner, the other reason are the usually longer intervals between iteration compared to methods like SCRUM. Furthermore, documentation cannot be deemphasized as it is a result for the funding bodies, and testing is often not sufficiently included in project effort and time planning. The reasons for this are manifold: the project goal is politically usually to do research, not spend time on product development. On the other hand, pilot applications are needed to prove project results. Resources are limited - as they always are - but to win a research project usually the innovative features are stressed, not the standard requirements for software in general (reliability, scalability, etc.). Therefore, time for testing is cut. On the other hand, some industry partners offer mature products to be incorporated in the research prototype. While these offer a high degree of maturity and usability, the research project may clash with the established development process of the product, causing for example compatibility and prioritization conflicts between new product and research features.

The lack of a central authority impacts development processes not only due to lack of a process owner, but also in terms of conflict management. Different views on requirements and effort estimates can often not be resolved by researchers, who strive to have a good working relationship, leading to a postponement mindset. The interviewees stress the importance of clearly defined development phases and their associated results. Multiple integration steps should be planned to avoid a so-called *big bang integration*, where all components are integrated at the same time (they usually aren't).

Additionally, the importance of a solid conflict resolution process is common between all reviewed projects. A single partner not achieving a development milestone may lead to issues for all other partners. Reasons for this may range from underestimated technical challenges and employee turnover to bankruptcy of a partner. In case of delays or partner

departure a common solution must be agreed upon to continue the project in a useful way.

5 SOFTWARE ARCHITECTURE

Conway's law states that "organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations" (Conway, 1968). In research projects, communication structures are characterized by geographical and organizational division. This limits the flow of information, not only because of increased effort to communicate (Herbsleb and Grinter, 1999a), but also because development artifacts like source code are not fully shared between all developers.

Thus, software architecture in research projects are bound by the following constraints:

- Software components need to be separated between project partners.
- Interfaces between components (and partners) need to be clearly defined.

In addition, the following constraints may apply to individual partners:

- Existing software components need to be reused
- Compliance to individual standards and best practices needs to be maintained
- Integration with existing software not relevant to the research project
- Software components need to be able to operate without components of project partners

5.1 Software Architecture Issues

When creating a software architecture within the aforementioned constraints, some issues arise and need to be addressed. In this section we describe issues from current and past research projects as well as methods used to address them.

Tool Compatibility

Resulting from existing best practices and standards, different tools may be used for the same task. While data formats, interface definitions, etc. are mostly compatible between modern IDEs, this is not the case for other tools.

One example are modeling tools, e.g. for UML. Common models are often used to arrive at a joint understanding of the research prototype, as well as its interfaces and components. However, if different modeling software tools are used, files may be incompatible, resulting in high effort to maintain common models or, if that effort is not undertaken, in outdated or fragmented models.

Selection of Standards

For purposes of interoperability, standards should be used within a research prototype. In distributed systems, important standards are communication protocols (e.g. SOAP or REST) and data formats (e.g. XML or JSON). Ideally, a common standard should be agreed upon between all project partners based on industry best practices, requirements and know-how. However, if the project partners have different best practices or want to use existing software for which design decisions are already made, finding common ground is hard, as each choice would mean considerable integration effort for at least one partner.

While it is possible to use adapters and enterprise application integration solutions (e.g. an enterprise service bus), these add additional expenditures for development and operation. If no partner is willing to provide a central solution, for example because no budget was allocated for this task, the only viable result is a point-to-point integration of software components, resulting in additional efforts for each partner and a suboptimal system architecture.

Data Management

Part of software architecture design is master data management, i.e. specifying how data is stored and which component is the principal system governing the data. Considering that partners may want their components to run standalone after the research project, this question has political implications. Software components relying on data storage from another partner cannot run without that partners' software. If each partner wants to maintain data sovereignty, the result may be fragmentation of master data. E.g. in the motivational example, bookings remain in the database of SV1, while the schedule containing the bookings remains in the database of UNI. Such fragmentation means redundancy in data, as different aspects of the same entity (e.g. a booking) are stored across multiple databases. This has far-ranging effects on the software, including consistency and traceability.

Transactions

Business activities should be transactional. However, as distributed research prototypes are loosely coupled and data is often distributed between different systems, transaction safety may not be guaranteed. This may lead to inconsistent data if the handoff between different partners' components fails. E.g. in the motivational example, a booking may be created in the Booking Frontend of SV1, but due to an error not scheduled by the Scheduling component of UNI. Therefore, a booking exists without being scheduled.

Without correct error handling this inconsistency may lead to incorrect behavior, e.g. the worker unexpectedly not getting a tool.

Considering that research prototypes are tested less than comparable software products, errors during operations are to be expected, which may lead to considerable efforts tracing and fixing data consistency errors.

Traceability

For locating and fixing errors, it is important to be able to trace the execution of software. In distributed systems, this presents unique challenges, as executing systems are heterogeneous, run in parallel and do not have a global state (Beschastnikh et al., 2016). In research projects, an additional hurdle is the organizational aspect. For each partner, the other partners' software is essentially a black box and cooperation is needed to trace bugs across components.

If existing systems are used, another hindrance may be the lack of global IDs for data items. For example, the ID of a booking may differ between the databases of SV1, SV2 and SV3. In this case, even identifying the same data item requires additional effort. To improve traceability, global IDs should be used for the same data item.

Similarly, the state of a data item is not easily ascertained in distributed systems. In the motivational example, to determine the state of a booking, the state data of the booking itself, its position in the schedule and the state of the booked tool need to be known, necessitating data from three project partners. If resources are limited, adequate administration frontends to view this data online may not exist.

Integration

Research projects operate under a tight schedule. Software must not only be created during the project timeline, but must also be used and evaluated to achieve actual research results. Thus, an early integration of components is critical to ensure components work together and are sufficient for pilot use. However, integration in distributed projects is also associated with high costs, as developers are in different locations and have different development and code styles (Herbsleb and Grinter, 1999b). Consequently, effort for integration is often underestimated. Combined with different levels of commitment by partners this may lead to only partially integrated systems, which do not support all features and have insufficient error handling.

Resilience

As components tend to be developed individually first, resilience is first handled on a component level. Error handling and recovery in the combined solution requires a high degree of coordination between components, e.g. for replaying process steps, rolling back updates, etc. As this coordination does not benefit project partners beyond the project scope and can only be implemented after the already time-constrained implementation of individual components, resilience of research prototypes tends to be low.

In the motivational example, for a tool booking to be successful, all four software components of the process must work in sequence. If an error occurs at any component, the whole process fails. As the state is distributed, recovery from such an error is not possible without cooperation between all project partners.

Some interviewees noted they added software components to compensate for issues the partners were unable or unwilling to fix, for example a caching system due to unexpected downtime and a plausibility check for calculation formula due to unhandled bugs.

Unclear Responsibilities

As described, each partner has individual goals and responsibilities in a research project. This in turn dictates the software components, a partner is responsible for. Due to time pressures during the proposal phase and the open-ended nature of research, often additional necessary components are identified during a project. If no effort was planned for such a component and it matches no partners' individual interest, there is no way to determine a responsible partner for the additional workload. In the motivational example, the need for a worker notification component was identified during the project, but it is unclear by which partner it will be implemented. Similarly, if responsibilities like error handling and input validation are not precisely defined for existing components, partners may omit such features in their components, leading to additional efforts by partners re-implementing this functionality in the requesting component.

While some of the challenges outlined above are inherent to the nature of distributed projects, others can be mitigated or avoided using organizational, technical and architectural solutions. Though all interviewees experienced some of the challenges, all were confident that software development in distributed projects can be successfully managed, provided the premises are taken into account. In particular, more experienced interviewees with more than one research project explained lessons they learned and applied in further research projects.

6 CONCLUSION AND OUTLOOK

In this work we described the dilemma of software architecture in distributed research projects - compo-

nents must work together in the context of the project but are built to function apart for later use. We described the characteristics of software development in distributed research projects and used them to show constraints and resulting challenges for software architecture. Software architecture is a critical success factor in software development. Researchers and developers should be aware of the unique architectural challenges in research projects as well as their and their partners' individual interests.

Based on the results of this position paper, we plan to investigate new and existing technical solutions and architectural patterns in regards to the presented challenges. In particular, integration methods in enterprise like Continuous Integration, Service-Oriented Architecture, REST, Enterprise Service Bus, and Microservices should be investigated in regards to their applicability for research prototypes.

While the high individuality of research projects makes a one-size-fits-all solution unlikely, a set of architectural patterns and best practices would contribute to ease development of research prototypes while still fulfilling individual requirements.

ACKNOWLEDGEMENTS

The authors would like to thank all interview participants.

REFERENCES

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al. (2001). Manifesto for agile software development.

Beschastnikh, I., Wang, P., Brun, Y., and Ernst, M. D. (2016). Debugging distributed systems. *Queue*, 14(2):50:91–50:110.

Conway, M. E. (1968). How do committees invent. *Datamation*, 14(4):28–31.

Derntl, M., Renzel, D., Nicolaescu, P., Koren, I., and Klamma, R. (2015). Distributed software engineering in collaborative research projects. In *2015 IEEE 10th International Conference on Global Software Engineering*, pages 105–109. IEEE.

Grinter, R. E., Herbsleb, J. D., and Perry, D. E. (1999). The geography of coordination: dealing with distance in r&d work. In *Proceedings of the international ACM SIGGROUP conference on Supporting group work*, pages 306–315. ACM.

Herbsleb, J. D. and Grinter, R. E. (1999a). Architectures, coordination, and distance: Conway's law and beyond. *IEEE software*, 16(5):63.

Herbsleb, J. D. and Grinter, R. E. (1999b). Splitting the organization and integrating the code: Conway's law revisited. In *Proceedings of the 21st international conference on Software engineering*, pages 85–95. ACM.

Howison, J. and Herbsleb, J. D. (2013). Sharing the spoils: incentives and collaboration in scientific software development. In *Proceedings of the 2013 CSCW conference*, pages 459–470.

Jacobson, I., Booch, G., Rumbaugh, J., Rumbaugh, J., and Booch, G. (1999). *The unified software development process*, volume 1. Addison-Wesley.

Lawrence, K. A. (2006). Walking the tightrope: The balancing acts of a large e-research project. *Computer Supported Cooperative Work (CSCW)*, 15(4):385–411.

Nguyen-Duc, A., Cruzes, D. S., and Conradi, R. (2015). The impact of global dispersion on coordination, team performance and software quality–a systematic literature review. *Information and Software Technology*, 57:277–294.

Powell, A., Piccoli, G., and Ives, B. (2004). Virtual teams: a review of current literature and directions for future research. *ACM Sigmis Database*, 35(1):6–36.

Spencer, D., Zimmerman, A., and Abramson, D. (2011). Special theme: Project management in e-science: Challenges and opportunities. *Computer Supported Cooperative Work (CSCW)*, 20(3):155–163.

Winkler, D., Mordinyi, R., and Biffl, S. (2013). Research prototypes versus products: lessons learned from software development processes in research projects. In *European Conference on Software Process Improvement*, pages 48–59. Springer.