Empirical Study of Tool Support in Highly Distributed Research Projects

Christian R. Prause
Fraunhofer FIT
Schloss Birlinghoven
53754 Sankt Augustin
christian.prause@fit.fraunhofer.de

René Reiners
Fraunhofer FIT
Schloss Birlinghoven
53754 Sankt Augustin
rene.reiners@fit.fraunhofer.de

Silviya Dencheva
Fraunhofer FIT
Schloss Birlinghoven
53754 Sankt Augustin
silviya.dencheva@fit.fraunhofer.de

***Abstract*—The EU subsidizes research projects in the ICT area with hundreds of millions of Euros per year with the aim of strengthening Europe's global competitiveness. A key requirement of EU projects is the involvement of partners from at least three different countries. This leads to highly distributed software environments where company, country, and culture boundaries run in the midst of tasks like requirements engineering, architectural design, implementation or testing. We present results from an empirical study involving more than 50 transnational, multi-million Euro projects of the Sixth Framework Programme. The results show which tools are accepted by developers and used in practice in the respective phases of the software process. Finally, we shape the idea of Research Software Engineering.**

I. Introduction

Based on the treaty establishing the European Union (EU), Framework Programmes serve two main strategic objectives[1]: (i.) to strengthen the scientific and technological bases of industry and (ii.) to encourage international competitiveness while promoting research activities in support of EU policies. One such framework is the Sixth Framework Programme (FP6), running from 2002 until 2010 with a total co-funding (EU pays about 75%) of 18 billion Euros. About 1000 projects with a total cost of about 5.5 billion Euros are co-funded in this way with about 4 billion Euros in the area of Information and Communication Technology (ICT).

The core of FP6 are collaborative projects that also account for the main funding. Collaborative projects are either large-scale Integrating Projects (IP) (funded with tens of millions of Euros) or Specific Targeted REsearch Projects (STREP) (several hundreds of thousands of Euros). A key requirement for funding is that projects include partners from at least three different countries. In practice, research consortia consist of partners from even more different countries. Similarly important is the involvement of both industrial and academic institutions, including universities, research institutes, small and medium-sized enterprises, large IT corporations, or even public bodies. We find from the survey in Section III that about 60% of our respondents consider themselves as academic partners, while almost 40% come from industry.

These research projects are a third kind of projects besides industrial and free open source projects that are nonetheless interesting to software engineering with an average cost of more than 5M Euros per project. Of course, compared to industry and open source projects, research projects are a bit different. EU projects share with industrial projects monetary factors but there is less pressure for software cost and quality. This is especially true for academic partners; to them software quality and cost reduction is secondary. Instead they aim for publications at high-reputation venues [12]. The projects' experimental character makes success and failure difficult to measure. Success is largely based on a review conducted once per year where the European Commission (EC) — as the executive branch of the EU — assesses if project progress seems reasonable and on schedule. Furthermore, there are no global competitors as EU projects are mostly restricted to European countries. Hence competition is not as fierce as in the free market. Therefore outsourcing and software process improvement are not equally necessary. EU projects share with open source projects a high degree of distributed development and volunteering to assume tasks. Yet developers might not be similarly motivated to develop good software [8].

Although EU projects involve academic research partners and have to generate scientific publications, they still tackle specific challenges and are required to create industrially exploitable solutions. For example, the Hydra Project[2] builds a middleware for embedded systems, is expected to set new standards, and generates an open source implementation. Industrially exploitable research results are a necessity to strengthen Europe's global competitiveness, and are some kind of Return on Investment (ROI) for the EU.

It is the nature of EU projects that company, country, culture and language boundaries run straight through the midst of a project task. It is not one partner that is responsible for requirements engineering but several. Similarly, several partners will do architecture design, implementation work, testing, and so on. This causes high communicational and, as project consortia build on flat hierarchies with friendly cooperation, also high self-organizational needs.

We ourselves have participated in several EU projects. Pro-

[1] http://cordis.europa.eu/fp6/fp6_glance.htm

[2] http://hydramiddleware.eu

This is the peer reviewed version of the following paper: C.R. Prause, R. Reiners, S. Dencheva. *Empirical Study of Tool Support in Highly Distributed Research Projects.* 5th International Conference on Global Software Engineer (ICGSE), Princeton, NJ, USA, pp. 23–32. See https://doi.org/10.1109/ICGSE.2010.13.. This version of the paper is posted here in accordance with the terms and conditions for use of self-archived versions. Self-archived at: https://www.drprause.de/publications/ICGSE2010-SoftEngPoll.pdf. BibTeX

cesses out of the schoolbook are difficult to implement sometimes. Even simple best practices like using the common Subversion repository to exchange code instead of distributing code via Skype can be hard to establish. A Subversion filter that rejects checking in of compilation results can make partners refuse to use it. Often there is almost no exchange of necessary documentation between partners, or it happens through outdated deliverables. And requesting partners to write Javadoc documentation can be considered overly bureaucracy by some. However, this is not malevolent behavior! It reflects that certain practices are just differently prioritized by different partners, or considered too clumsy or just unsuitable.

However, other approaches — like a Jira[3]-based Volere process for distributed requirements engineering — were successfully used in such environments [16]. In general, processes are the result of consensus in a consortium of equitable partners and not established by force of a single higher instance.

We became curious about the general situation in such projects (see Section II) and how they deal with the different tasks of a distributed software life cycle. We analyze research projects by surveying their tool usage through a questionnaire. Additionally, we capture developers' opinions on the usefulness of respective kinds of tools (see Section III). Although the validity of our survey is exposed to some threats (see Section IV), we discuss lessons learned from a valuable first glance at practices of a multi-billion Euro research "industry" that has been widely omitted from software engineering research (see Section V). But as well as research can learn from industrial rigor in distributed development, industry can learn from a community that has a large amount of experience in extremely distributed software development (see Section VI). Therefore, we propose that methods from industry projects will be taken into account and modified to suit the needs of research projects (see Section VII). The survey presented in the following sections will help us to find this out.

II. Survey Scope and Realization

According to the IEEE Standard for Software Life Cycle Processes [2], software development entails a variety of processes that are categorized in groups like Project Management, Post-development, etc. However, not all processes are applicable or mandatory for the execution of a software project. We focus on processes of the "Development" process group but also include a few processes from other groups.

Relevant to our study are tools that support management, collaboration and communication, quality improvement or other key tasks of the software life cycle. For example, managing requirements can be considered a key process in all software projects, and usually there is a single tool that supports this process. Yet for other processes such exclusivity is not given: the Implementation process is usually supported by a variety of tools like integrated development environments (IDE), profilers, debuggers, programming languages and so on. It would have been an impractically tedious task to ask developers about all facets of their development environment. Instead, we chose to leave out tools that support the individual developer only in this survey as they do not directly affect the distributed groups' workflows.

Similarly, we decided that tools should be broadly applicable to software development, and not be domain-specific or serve a small niche. Apache Axis, for example, includes a tool that generates WebService source code in Java from Web Services Description Language (WSDL) files and simplifies software development, but it is specific to the WebService-domain. Likewise, model checkers address a niche where extreme reliability is required. So we decided to exclude such tools. But still we include static analysis tools (see Section III-B) because they can be used in the project and collaboration context (apart from personal IDEs), for example, in continuous integration servers to promote good coding styles [3].

The rest of this section deals with the conduction of the survey, explaining how projects and interviewees were selected, and what the different phases of the survey were.

A. Project and Participant Scope

Besides the FP6 the EU finances several project programs. Partly these other programs run in parallel to the current Framework Programme. One example is the eContentPlus[4] program. But the Framework Programmes are also a series of programs that are natural successors of one another.

We chose FP6 (as opposed to its successor FP7) because the projects funded in this framework are already finished or have at least reached a high degree of maturity. Additionally, as no new projects will be funded in FP6, their total number is known and remains constant. This is not the case with FP7 where the first projects started recently in 2009. Many projects are in their early phases and it is still open, how many and what other projects will be funded. Furthermore, newly started projects might not yet have agreed upon all software engineering process implementations and their tool support.

We opted against FP5 (the predecessor of FP6, running from 1998 until 2002[5]) because all its projects ended years ago, and many people involved in these projects have probably changed their affiliations since then; this is even more critical for academic institutions where personnel often has short-time contracts. Furthermore, tools available in the year 2000 might not be state of the art anymore; vice-versa today's tools might even not have existed at that time.

B. Survey Phase One: Addressing Coordinators

Three phases make up the conduction of our survey: contacting coordinators, inviting participants, and finally evaluating the results. From February 9, 2009 until July 2, 2009 we sent emails to the coordinators of the 1201 project listed in the CORDIS[6] (Community Research and Development Information Service) database. CORDIS contains information on European innovation and development provided directly from

[3]http://www.atlassian.com/software/jira/

[4]http://ec.europa.eu/information_society/activities/econtentplus/

[5]http://cordis.europa.eu/fp5/

[6]http://cordis.europa.eu/home_de.html

the EC. Our email contained information about the planned survey, its contents and goals, and stated that we are working for the fellow EU project Hydra. A preliminary version of the questionnaire was attached for preview. We reassured to our participants that no results for a specific project would be evaluated and presented. The coordinators were asked to assess, if their project is a software project with respect to the study, and if so, to name five different persons deeply involved in software development. Coordinators were asked to give feedback about the nature of their project in any case.

During this process, it turned out that CORDIS contained 34 duplicate entries for projects so that the total number of projects is actually 1167. For 79 projects no contact information form was available, four more projects were associated with invalid information. We do not know how many coordinators actually received our mails, as Hydra's coordinator, for example, went out of business a year before this study. They were still listed in the database, yet contacting them experimentally through CORDIS did not return an error. Furthermore, some of the projects had ended a few years before our survey, so it is uncertain if the coordinator was still contactable through CORDIS. In the end, the number of project coordinators addressed was 1084 but the number of coordinators actually reached may be lower.

Out of these, 115 coordinators responded. As this number exceeds one tenth of the 1084 contactable unique projects, we consider following assumption as being representative: 42 coordinators stated that no software was developed within the project (e.g. Networks of Excellence or chip-design projects) or that there was no software which would make sense in the scope of the questionnaire (e.g. only simulations in Matlab). 73 coordinators responded that within their project, software was developed. That means only approximately 63.5% of 1167 projects are developing software. Consequently, for our evaluation we assume a basis of 741 relevant projects.

C. Survey Phase Two: Inviting Participants

After the initial phase, we either immediately received emails from the coordinators containing contact data of five developers, or we received emails from volunteering project members themselves as the coordinators had refrained from giving out contact information, but instead circulated our contact mail. This resulted in a high participation rate later on.

The personal invitations to individual online questionnaires were mostly sent between August 18 and August 20, 2009 to avoid sending invitations on Mondays and Fridays as we deemed these days unsuitable for answering a questionnaire. Some of the emails of possible participants had become invalid in the mean time. However, their number is negligible. Additionally, our questionnaire ended with an input field where further candidates for the survey could be proposed. This resulted in about 10% more responses.

D. Survey Phase Three: Data Refinement and Evaluation

After receiving the results which are presented in Section III, typos and abbreviations entered into the optional free text boxes had to be corrected and generalized. For instance, "svn" was mapped to "Subversion" or "internal tools" to "custom". Where possible, some few simplifications ("JUnit+Clover" became "JUnit/NUnit") were also made in order to facilitate the categorization of answers. Since we analyze general trends and opinions, this generalization is considered as valid.

The questionnaire asked participants about their use of tools locally in their group, and the use project-wide. By design of the survey, it was possible to receive more than one response per project. However, in some answers provided to the "project-wide" questions, we encountered inconsistencies that were difficult to resolve: for example, participants named different tools supporting version control. It actually happened that one participant answered "Subversion", another one "None", and a third one "CVS". We assume from an in dubio contra reo point of view that the answers are guesses; therefore "None" was set. But it is also possible that the one answering "None" is just not aware of the tool, while one of the others just errs in the kind of tool. Hence, a "None" answer can mean "don't know", whereas a "don't know" can be as bad as a "None" because a project-wide tool is useless if it is widely unknown and not used.

In order to handle these inconsistencies, we derived a rule specifying the *consistency factor* of given answers within the same project. This means, that the first answer is taken but does not influence the consistency factor. Every following answer either confirms or confutes the first answer. Depending on this rule, the consistency or inconsistency of the answers is influenced. This rule was applied to all questions in order to derive the consistency factor for the whole evaluation of all questionnaires within one project. The overall answer consistency factor is 59.7%; the number of consistencies ($n > 300$) divided by the sum of consistent plus inconsistent answers.

In the evaluation each project is represented with the same weight, no matter if one, two, three or more persons answered for the same project. So, one response among several ones for a project would then have proportionally less weight, e.g. one half or one third. This method seemed to us the only fair resolution to the problem.

III. The Questionnaire

In this section we provide a detailed description of the questionnaire and the answers we received. The presentation follows the original order of the questions.

In total, we sent out 243 questionnaires. Out of these, 91 were completely ignored and 46 were only partially completed. But 106 questionnaires (44%) were completed.

Questionnaires for 54 different projects (approx. 7.3% of all software projects) were completed. The participating countries were Germany (29.2%), Italy (10.4%), Spain (6.6%), Austria (5.7%), United Kingdom (5.7%), the Netherlands, Slovakia, Israel, Switzerland, France, Poland, Greece, Hungary, United States, Finland, Belgium, Sweden, Lithuania, Czech Republic, Canada, Cyprus, Denmark, and Norway. This list contains a few non-EU countries that nonetheless supplied partners thus giving EU projects a global dimension. We think that the high

percentage of answers from Germany results from national and corporate (Fraunhofer) togetherness, a generally high portion of Germans in this field, and from contacts on a direct and personal level. This is supported by the observation that several people from other countries reacted to the invitation, but did not finish the questionnaire.

A. Structure of the Questions

A short sentence introduced each question. The questions were either single- or multiple-choice with examples of typical tools in that area as to clarify what kind of answer is expected. For example, for requirements management we presented a list containing "Rational RequisitePro", "Atlassian Jira" and others. Questions typically offered an additional free-form field into which participants could enter any other tool or a combination of tools they used.

Most questions consisted of two parts: One asking for what approaches and tools participants use in their local workgroups for the respective projects — the other one asking for what is used project-wide. We added a special "`Provided by project`" option to test for collaboration in the project: Participants should select this option if there is a centralized tool that is provided by the project, and that is used by the entire consortium. But if a participant's group and the whole project use the same tool indeed but different installations then they should not select this option but provide the same answer to both questions.

Additionally, for most questions participants were asked to share their personal opinion with us, and rate the usefulness of the respective kind of tool.

B. Details on Questions and Answers

This section lists all questions with a short summary of the answers. The answers are presented according to the agreements in Section II-D in tables accompanying the questions. For each table, only the five mostly chosen answers are listed. The rest of the options is summarized to "others" and is textually described when considered as appropriate. Next to the list of the most favorite tools, personal views regarding the usefulness of this kind of tools ranging are shown. Possible values range from 1 (not useful) to 5 (absolutely useful).

Q1 - Project Management: Tool supported project management is done a lot in research projects (cf. Table I). This holds especially true for project-wide project management, where only a few percents of the projects do not have project management. The researcher's opinions are mainly positive towards project management. Yet, project management is often done using office applications (including OpenOffice).

Microsoft tools dominate this field with MS Project being the mostly used dedicated project management tool. Some groups/projects additionally rely on SharePoint. But other tools like ACollab, Google Calendar, diverse issue trackers, or even custom applications are used, too.

	Grp.	Proj.
MS Office	27%	27%
MS Project	25%	22%
Office + Visio	10%	5%
Jira	3%	3%
Other	13%	15%
Don't know	2%	20%
None	11%	12%
Prov. by proj.	13%	
Σ	106	54

	Grp.	Proj.
1	3%	5%
2	10%	18%
3	40%	33%
4	37%	37%
5	10%	7%
Δ	8	22
Σ	98	84

1=Not, 2=Few, 3=Fair, 4=Good, 5=Absolutely

TABLE I: Project Management & Opinions

	Grp.	Proj.
MS Office	7%	8%
Jira	3%	4%
eRoom	3%	2%
DOORS	2%	0%
Other	6%	4%
Don't know	10%	23%
None	58%	60%
Prov. by proj.	14%	1%
Σ	106	54

	Grp.	Proj.
1	11%	21%
2	23%	21%
3	34%	36%
4	24%	17%
5	7%	5%
Δ	36	40
Σ	70	66

1=Not, 2=Few, 3=Fair, 4=Good, 5=Absolutely

TABLE II: Requirements Management & Opinions

Q2 - Requirements Management: Requirements management plays a vital role in many software processes, be it classical or agile ones. Requirements justify the software development in commercial projects. However, its value for research projects is disputed because it is not clear in advance where the research is going to. But requirements capture decision rationale that is indispensable for reuse. Results and opinions on tool-supported requirements management in workgroups and project-wide are shown in Table II.

Many EU research projects are subsidized but no concrete goals are defined. The project Hydra, for example, has a sophisticated requirements process [16], and is very successful, yet development is not that requirements-driven as intended in the beginning.

Tool supported requirements management is done rarely but is very often judged with at least "fair" usefulness by more than half of the survey participants. It is a candidate where actual use needs to catch up on expectations. Other tools used by research project include DOORS, Rational RequisitePro, office documents, and bug trackers.

Q3 - Architectural Design: Creating the architectural design of software is the task of software architects who frame the general design of software before it gets into the implementation phase. It is the typical domain of modeling languages like UML. This is also reflected in the results of our survey where we asked what tools are used by researchers in order to create their architectural design specifications in their groups and project-wide, respectively (cf. Table III).

Besides dedicated architecture tools like Enterprise Architect or Rational Architect, plain office and LaTeX documents in combination with graphs from Visio (and a host of other UML tools like Poseidon, UMLet, etc.) are used in almost all

	Grp.	Proj.
Office + Visio	35%	25%
MS Office	26%	30%
Entpr. Architect	13%	10%
Rational Architect	5%	0%
Other	12%	9%
Don't know	2%	25%
None	3%	4%
Prov. by proj.	7%	
Σ	106	54

	Grp.	Proj.
1	1%	6%
2	6%	8%
3	40%	38%
4	34%	35%
5	19%	13%
Δ	4	21
Σ	102	85

1=Not, 2=Few, 3=Fair, 4=Good, 5=Absolutely

TABLE III: Create Design & Opinions

	Use
Personal email	87%
Mailinglists	71%
Telephone	53%
Meetings	44%
Instant Mess.	33%
Wiki	19%
Video confer.	13%
Skype	5%
Σ	106

	Grp.	Proj.
Email	61%	67%
Deliverables	35%	61%
Wiki	29%	32%
BSCW	17%	26%
Shared drive	29%	13%
Subversion	12%	11%
Don't know	0%	2%
Prov. by proj.	32%	
Σ	106	54

TABLE IV: Communications & Document Sharing

	Grp.	Proj.
Subversion	58%	42%
CVS	13%	6%
SourceSafe	3%	0%
GForge	0%	2%
Other	4%	4%
Don't know	2%	21%
None	14%	27%
Prov. by proj.	9%	
Σ	106	54

	Grp.	Proj.
1	0%	4%
2	5%	10%
3	8%	17%
4	44%	44%
5	43%	26%
Δ	10	28
Σ	96	78

1=Not, 2=Few, 3=Fair, 4=Good, 5=Absolutely

TABLE V: Version Control Tools & Opinions

research projects to create architecture design specifications. More exotic approaches are white-board drawings in combination with digital cameras for distributing designs to remote teams.

Q4 - Documentation and Communication:
(a) Sharing of Documentation and Specifications Documentation and specification of software architectures and its components is an important part of the software engineering process [19]. In a distributed project with different sub-teams working under one "common roof", up-to-date documents are necessary. Table IV shows what kinds of tools are used in order to share documentation and to keep it up to date. It also presents an overview of the tools used for that purpose. Several answers were allowed since normally, more than one tool is used or different ones are combined.

From the results we see that email communication is widely used, as well within one group as throughout the whole project. It seems to be accepted to send plain text information together with eventually attached specifications. The resulting personal overhead and obvious disadvantages for every recipient seem to be tolerated. Deliverables are another important kind of documenting mechanism, though there is the danger that these polished documents are not alive and easily grow out of date.

More collaboration-oriented tools like Wikis and shared workspaces (e.g. BSCW) or network drives are quite prominent in groups and projects. Also version control tools like Subversion are used to organize, update and distribute shared documentation.

(b) Typical Means of Communication: When asking for documentation and sharing mechanisms, we regarded the personal communication within a project as a linked aspect to documentation. Communication is the basis of collaboration. In order to be able to agree on architectures and strategies, personal communication via audiovisual channels or text exchange is vital. We asked how communication during the project progresses happened and what tools were used for that. For the answer, several options could be chosen but the participants were asked to only choose tools they use at least once a week.

Most participants rely on email and mailing list communication (cf. Table IV). After these often named options came phone calls and personal meetings. Only one third harnesses instant messaging for spontaneous and in-between communication though this is a key factor to project success in distributed projects [13]. Wikis and video conferencing are also named but their usage is rather uncommon.

Q5 - Software Configuration Management: Originating from a US military standard of the 1960s, software configuration management is one of the oldest and most widely accepted software engineering tools, even older than software engineering itself. Opinions about and results for managing software configuration items (especially source code) with version control systems can be found in Table V.

There is full agreement on the usefulness of this kind of tools that is supported by high actual use ratios. Even project-wide there is a high agreement that version control should be used. However, there is a low "provided by project" ratio suggesting that there is no common version control but lots of local sub-projects.

Though free and centralized version control systems hugely dominate, distributed or commercial ones like Git, Mercurial or Plastic SCM are also used. Some groups/projects use BSCW or FTP folders for their configuration management.

Q6 - Test Plan and Test Management: Test plans document and describe the testing approach including the tests to be performed as well as the expected results [1]. Some tools, like HP Quality Center, help managing the interplay between testers and developers, and generate reports.

See Table VI for detailed results on how test plans are managed, and personal opinions about test plans. Mostly simple text files are used in almost one third of the projects to specify and maintain a test plan. This was stated for groups as well as projects. None of the groups/projects had a standards conformant test plan. Spread sheets are the next often used alterna-

	Grp.	Proj.
Simple text	33%	23%
spread sheet	32%	22%
custom	3%	0%
eRoom	1%	1%
Working examples	0%	1%
Other	4%	2%
Don't know	3%	25%
None	27%	28%
Σ	106	54

	Grp.	Proj.
1	8%	13%
2	20%	12%
3	41%	46%
4	22%	22%
5	9%	7%
Δ	16	30
Σ	90	76

1=Not, 2=Few, 3=Fair, 4=Good, 5=Absolutely

TABLE VI: Test Plan & Opinions

	Grp.	Proj.
Hudson	4%	4%
CruiseControl	3%	3%
Team Found. Server	3%	1%
QuickBuild	2%	2%
Other	3%	4%
Don't know	6%	19%
None	78%	71%
Prov. by proj.	3%	
Σ	106	54

	Grp.	Proj.
1	27%	35%
2	18%	18%
3	21%	33%
4	27%	9%
5	6%	5%
Δ	44	51
Σ	62	55

1=Not, 2=Few, 3=Fair, 4=Good, 5=Absolutely

TABLE VIII: Continuous Integration & Opinions

	Grp.	Proj.
JUnit/NUnit	34%	20%
custom	2%	0%
Hopper	1%	0%
Visual Studio	1%	0%
unknown	0%	1%
Other	4%	1%
Don't know	0%	5%
None	60%	75%
Σ	106	54

	Grp.	Proj.
1	16%	30%
2	7%	13%
3	29%	28%
4	29%	21%
5	19%	8%
Δ	33	45
Σ	73	61

1=Not, 2=Few, 3=Fair, 4=Good, 5=Absolutely

TABLE VII: Unit Testing Tools & Opinions

	Grp.	Proj.
Meetings, no tool	51%	31%
Jupiter for Eclipse	2%	0%
Visual Studio	1%	0%
Git	1%	0%
Other	2%	0%
Don't know	3%	16%
None	38%	55%
Prov. by proj.	3%	
Σ	106	54

	Grp.	Proj.
1	9%	21%
2	16%	26%
3	47%	33%
4	21%	14%
5	7%	6%
Δ	25	40
Σ	81	66

1=Not, 2=Few, 3=Fair, 4=Good, 5=Absolutely

TABLE IX: Code Reviews & Opinions

tive — again concerning groups and project-wide agreements. 26% and 28%, respectively, responded that there was no test plan or management. Besides above alternatives, some groups and projects make use of Mantis, SiTEMPPO or Testlink.

Q7 - Automated Unit Testing: This question aims at finding out whether unit testing plays a role in software development at all. We defined that test coverage should be at at least 20%, to ensure that we only capture serious testing. Otherwise, "None" should be chosen as answer. Results and opinions concerning unit testing are shown in Table VII.

Half of the groups stated that no testing is applied; 75% for projects. Those groups and projects that do testing almost only rely upon JUnit or NUnit tests. A small percentage uses other tools like GoogleTests or Yaffut.

Within groups and projects more than two thirds rate the usefulness of unit testing at least "fair", though there are some deniers. There is a basic acceptance for testing.

Q8 - Continuous Integration: Continuous integration originates from Extreme Programming to support agile software development. An dedicated server regularly (e.g. nightly) builds the software in a clean environment and automatically runs tests. Hence, version control, automated build scripts and unit tests are necessary. This implies additional work for server setup and writing non-IDE build scripts, but gives confidence that submitted code always compiles, and that errors are quickly detected and therefore cheap to fix [17].

Table VIII shows that the general acceptance of continuous integration is very low, especially in groups. Almost three quarters state that within groups and project-wide, continuous integration plays no role. Maybe the cost for server setup and prerequisites like unit tests are considered as too much overhead.

Q9 - Code Reviews: Compared to other quality assurance measures code reviews are considered rather cheap. Yet they are capable of detecting a wide range of software defects, especially those for which a deeper understanding of the software is necessary, and hence are well-known in industrial and open source software development [18].

The answers to our questions (see Table IX) show that the acceptance of reviewing is high (especially in groups). Note that this time we did not ask for tools but whether reviews are executed at all; multiple answers to this question are possible. Half of the developers engaged in EU research projects do code reviews, and many consider reviewing fairly useful. But besides *Jupiter for Eclipse*, tool support is limited to text documents and web forms. Poor tool usage makes cross-partner reviews difficult and hence this option — although being a proposed answer — was not selected.

Q10 - Bug Tracking: Bug tracking means to manage encountered software errors and to keep track of the system's current status. Though it may be more practical when dealing with customers in big projects in which communication paths are complicated or the time-to-fix is long, it is still considered a very pragmatic process to follow [17].

Opinions on bug tracking, and results for groups and projects are shown in Table X. Many research projects do not establish error management processes; bugs are tracked via email communication. Within groups, the need for this kind of tools is considered as more important. Less commonly used other tools are Redmine, eRoom, Rational Jazz or GForge.

Q11 - Static Code Analysis: Again, several answers could

	Grp.	Proj.
MS Office	12%	10%
BugZilla	13%	9%
Trac	16%	5%
Jira	6%	7%
Other	13%	7%
Don't know	3%	19%
None	34%	46%
Prov. by proj.	8%	
Σ	106	54

	Grp.	Proj.
1	8%	14%
2	8%	16%
3	26%	32%
4	42%	29%
5	15%	9%
Δ	22	30
Σ	84	76

1=Not, 2=Few, 3=Fair, 4=Good, 5=Absolutely

TABLE X: Bug Tracking Tools & Opinions

	Grp.	Proj.
Code coverage	14%	9%
Lines of Code	16%	6%
FPA	11%	7%
Cycl. compl.	4%	1%
Static P. Count	1%	0%
siMetrics	1%	0%
None	58%	54%
Prov. by proj.	10%	
Σ	106	54

	Grp.	Proj.
1	22%	30%
2	16%	19%
3	38%	35%
4	21%	13%
5	3%	4%
Δ	33	52
Σ	73	54

1=Not, 2=Few, 3=Fair, 4=Good, 5=Absolutely

TABLE XII: Software Metrics & Opinions

	Grp.	Proj.
IDE built-in	60%	38%
CheckStyle	4%	3%
FX / Style cop	1%	2%
PMD	2%	1%
DevPartner Studio	2%	0%
NetBeans	1%	1%
None	42%	42%
Prov. by proj.	3%	
Σ	106	54

	Grp.	Proj.
1	8%	20%
2	14%	13%
3	33%	35%
4	33%	22%
5	13%	10%
Δ	20	46
Σ	86	60

1=Not, 2=Few, 3=Fair, 4=Good, 5=Absolutely

TABLE XI: Static Code Analysis Tools & Opinions

	Groups	Projects
Less than 5	64%	5%
Less than 10	33%	30%
Less than 20	4%	24%
Less than 40	1%	34%
40 or more	0%	9%
Σ	106	106

TABLE XIII: Code Contributors

be given. Static code analysis supports the developer while writing code through finding syntactical or low-level semantic errors in code without executing the code. Modern IDEs have built-in static analysis features and often suggest solutions to detected problems in tooltips. The results are shown in Table XI (The table shows how many groups use a respective tool; some use the IDE built-in functionality and additionally some other tool).

Surprisingly, there is a high rate of "None" answers. Almost every second group does not use any kind of static analysis. Either participants ignore their IDEs support or they misunderstood the question, because they regard the built-in methods as state-of-the-art and wonder what other tools could be used. As nobody selected "don't know", everybody seems to have a notion of code analysis. But even more developers profit from IDE built-in analysis. The high acceptance for this is illustrated by the opinions, although for an effort-free feature, rejection is quite high.

Q12 - Software Metrics: Measurement plays an important role in engineering and quality assurance. Though metrics are not error-proof, they are the basis for objective evaluation methods, and iterative improvement [11].

The results for this multiple-choice question are shown in Table XII. The general opinion is that half of the participants regard metrics as not useful for these projects. The numbers for groups and projects are quite similar. Also concerning the application of these metrics, more than half of the participants stated that it is not used; both in groups and project-wide.

Q13 - Code Reuse: Planning to reuse code is an important argument for code quality. Code quality, conversely, is an important argument for reusing code. Hence, it is important to know how research deals with the problem of code reuse. Participants should indicate how much software from a previous project they reuse and if they are planning to reuse code from the current project in future ones.

Participants chose between several discrete estimation values from 0% to 100%. The results show that (in the average) participants are reusing 27% of the code of former projects within their group, and 27% project-wide.

In the future they are planning to reuse 52% of the code within the group and 47% of the entire project. This is more than the actual reuse of old code.

Q14 - Number of Contributors: Local developer groups at a partner's site are rather small. They often have no more than 5 contributors, and rarely more than 10. However, EU projects are often bigger having up to 40 developers and sometimes even more than that (see Table XIII). Concerning the professional background of the participants in our study, most of them are applied researchers from academia (cf. Table XIV). The second largest group is represented by Industry Research & Development departments. Participants from fundamental research and industrial Products & Sales are a minority.

C. Participants Remarks

In this section we sum up on selected participant remarks that were submitted along with the questionnaires.

	Affiliation role
Academic - Fundamental Research	7%
Academic - Applied Research	55%
Industry - Research & Development	36%
Industry - Products & Sales	2%
Don't know	1%
Σ	106

TABLE XIV: Participant Role

The projects under study are, of course, research projects. A few participants wrote a remark along the lines that they only develop for researching and prototyping. The concepts are what they want to reuse, and not the software. Likewise, they do not deliver their results to customers, and therefore argue that software engineering best practices are not applicable for them. We are aware of the fact that software is rather experimental and not delivered to customers but do not agree that a multi-million Euro project builds software only to throw it away.

The team size is another reason why full-scale processes are considered inappropriate, e.g. if there are only two programmers sitting in adjacent rooms while working on implementations. By some, tool support besides a common CVS is considered a waste of efforts. Those argue that it was better to invest in building the actual software than to spend efforts on setting up processes. Justifications for (perceivably few) own usage of tools are that the developments are small and therefore easy to understand and manage. Or, less seriously, extreme quality targets that make bug trackers superfluous: one respondent answered that they cannot afford to track bugs because instead their philosophy is to fix them immediately when they occur. Other projects struggle with the fact that each group applies a different development methodology. In general, many processes are simplified and poorly extend to collaborating groups.

One suggestion was to concentrate on industrial projects instead as studying tool usage here may be disappointing. However, it was exactly our intention to study this kind of projects and this is the reason for this survey.

Service-oriented architectures (SOA) were also quoted a few times as a reason for having only a few integrated processes. The different subsystems of a SOA application are deployed to different machines and then connected through network calls. Every team can hence develop its parts more or less in isolation, and only needs to communicate interfaces to the others. From our own experience, however, we can tell that such organic development (without a clear architecture) tends to build a software system that is also quite confusing. It is a fallacy to think that a new implementation technology can solve all the problems of the software life cycle.

IV. Threats to Validity

There are many research projects which unfortunately did not responded to our invitations. Reasons for not answering are possibly low interest in software engineering, or being overwhelmed by the spectrum of best practice tools presented in our questionnaire. Yet, seven out of eight people who started their questionnaires filled it out completely. Hence, we think it is plain disinterest in participating in a survey. Still we regard the 106 responses for almost 10% of the projects as significant.

Participants in the roles of applied researches and researchers from industry are the vast majority of people in this study. We think the reason for this is that these are the two groups of people that mainly do EU research projects. They are therefore not severely overrepresented.

The response consistency factor of project-wide responses is below 100% (see Section II-D). This indicates some insecurity in the responses. Still the majority of answers is consistent and we gave a very tight definition of consistency.

A limitation of this study is that tool usage is vaguely defined, and does not guarantee quality. If, for example, there is a common Subversion repository in the project, it does not mean that it is actively used. Similarly, even if the tool is actively used, it does not mean that it is used correctly. But for an empirical study that wants to draw a broad picture this is still acceptable.

Another danger is that often many small developments are made within the individual groups of the consortium while our questionnaire covers many aspects of full-size software projects. This is due to technical heterogeneity in sub-tasks. In the scope of the project these implementations are then integrated later. Here integrated development with the participation of all partners is hard to realize.

Additionally, some research projects are very heterogeneous: a common, project-wide implementation is not always intended. Service oriented architectures allow widely autonomous development of software parts. Other projects are concerned with the development of embedded firmwares that have to be small and energy-efficient, or are purely optimization-oriented which makes standard tool support oversized for the scaled down processes. Sometimes, a small team first develops the software and then forwards it for integration to key researches actually working for the project's vision. Direct contribution within the project is therefore not possible. However, the number of such projects is small and we invited only software development projects. Several projects opted out or were rejected in the first phase of our study.

So, because of the diversity in research projects, partners and goals, it is difficult to make generally valid and all-encompassing assumptions. However, we enabled a valid insight into research practice.

V. Lessons Learned

Analyzing the survey results, we summarize that the backbone of projects is mostly considered as important and applied within the projects' landscape. These namely are project management (Q1), architectural design (Q3) and code management tools like SCM (Q6). Shared tools, and communication and information infrastructures are set up and used regularly.

When it comes to documentation and communication besides project management issues, the answers revealed that lots of communication takes place via email (Q4a/b). However, pieces of documentation and specification are very often sent via this channel, too. Similarly, there is astonishingly few requirements management (Q2) that could capture concept and design rationale. This leads to scattered information and causes organizational overhead for recovering and (re)organizing the information later (for example, an attempt to automate this process for an EU project was [14]). About one quarter of the projects already provide the technical infrastructure to collab-

orate and exchange knowledge in a managed way, for example via the BSCW system or administered Wikis.

Considering test plans and the management of tests (Q6), automated unit testing (Q7) and continuous integration (Q8), the collected answers reveal that these aspects of quality assurance and bug fixing are missing in many projects, or processes are set up but not maintained and applied consequently.

Code reviews and managed issue tracking (Q9 an Q10) are both considered as important, especially within the projects' sub-groups. Consequent tool-supported processes, however, are still lacking. The same applies to coding rules (Q13). Although considered as helpful and supportive, development goes on in various individual styles that are hard to combine, maintain and refactor for future purposes.

As consequence from the results, especially concerning Q4, Q6, Q7-Q10 and Q13 we see a difference from large-scale sales-oriented industry projects in terms of management, coercion and constraints. We make this assumption due to the highly distributed nature of research projects and the lower pressure from project management which only has limited power over partners that want to preserve their independence. Each change in regulations or organizations up to project demands and duties is very often based upon new negotiation where to much pressure from the management side may result in demotivation, frustration and finally worse results.

From the study we learn that even agile methods are rejected as heavy-weight, e.g. continuous integration or test-driven development. Coding rules are accepted and wanted, but enforcement is missing (which is considered even worse [7]). The discrepancy between expected reuse of software (47%/52%) and actual reuse (27%) shows that a desire for improvement is there. Possibly there is a discipline problem in EU projects that could be addressed with software process improvement programs or maturity models like CMMI; but may be there is also a need for adapted or all-new tools.

Generally, group-wide solutions are more accepted than project-wide solutions. Project-wide solutions are nonetheless mostly regarded useful. Their implementation as indicated by the "Provided by project" responses is very low, often around 10% or lower. This shows that project-wide processes have yet to be established, although appropriate tool support for software processes is extremely important in a distributed setting to reduce the effects of hampered communication, control and coordination [5]. However, we are aware that bringing together two different company's processes is a complicated and time consuming task on its own (e.g. [20]), especially if it is only for the duration of a single project of a few years.

Naturally, the goals of research projects are a bit different from conventional software projects. But from the study results and participant remarks we register that another important difference is how researchers perceive themselves and their projects: there is some sympathy for software engineering methodology but the cost for establishing and maintaining according processes is too high for a community that very much appreciates its freedom and equality. Any methodology or process that is applied top-down risks to be rejected as cumbersome, and there is no superior force that could help overcome initial resistance, e.g. for bug tracking. Acceptance for anything can only come bottom-up.

Solutions should consider the size of projects (number of participants), existing tool support but also the need for increasing the individual partners' motivation on community-based levels. In our experience, personal, friendly and sometimes patient conversation tones lead to more success and better contributions than putting pressure on partners. Future research has to focus on approaches that lead to higher individual motivations (e.g. incentives) to follow certain processes, and to increase the quality of software as well as documentation and communication. All this cannot be achieved by force.

VI. Summary

This work describes a survey among participants of EU research projects. Several persons from research projects of FP6 were invited to an online questionnaire that interviewed them on their use of tools and their personal opinions on the usefulness of that kind of tools. The survey shows what methods are applied in this large software field. Also opinions and the overall acceptance for such methods is presented.

We acknowledge that research projects have some differences compared to industry and open source. From our participants' remarks we see that many share this assumption. But research projects also share properties with industry and open source. For example, research projects receive funding from external bodies and are responsible to them; they must carefully plan and manage their resources. With open source projects, research shares freedom and intrinsic motivation through research interests. Friendly cooperation and voluntary contribution are key features. Both, EU and open source projects, have a high degree of spatial distribution, which shown to negatively impact process maturity and therefore software quality [5], [6].

In research projects, software quality and reuse are considered lesser goals. Nobody expects flawless operation from a prototype, but novelty. Collaboration and gain of new knowledge are important. However, we argue that quality has its space there, too. Reuse and some degree of quality increase productivity, which is interesting for research projects as well.

Many groups and projects plan to reuse code in the future, but actual reuse is low. We argue that there might be a discipline problem that is due to the special nature of research projects and cannot be addressed with standard industrial tools. Similarly, workflows are not suited to EU project needs. However, tools for design, documentation & communication, version control and coding rules are considered as useful, though they need to be woven into project life cycles more closely.

We are convinced that software development in EU projects can profit from tools and processes that are tailored to this kind of distributed software development. Increasing the reusability of software results from such research projects means to shorten the time-to-market considerably, which is of paramount importance for European competitiveness; failing to do so means to lose economic opportunities. Therefore,

our claim is that well-suited processes and tool support increases communication within the consortium, flexibility of the research process, and quality of developed software process artifacts. Only this enables efficient exploitation of results, and gives EU the much needed return on investment.

One might be tempted to shrug off the image of distributed software development drawn by this study because it was *only* research. But this would do the study wrong: it presents a view of today's distributed development practice as it includes a broad variety of software organizations. Reflection is the first step towards the improvement of a situation, and it shows which software engineering ideas and tools can survive in the development wilderness outside of large companies. This survival in different contexts is a key component in the evolution of software engineering ideas [10]. EU projects and global software development have many aspects in common.

Since findings are based on EU projects, we focus on problems that occur from spatial distribution of partners. Although socio-cultural, distance and time differences are smaller than in global development, they cannot be neglected wholly.

VII. Perspective

As a consequence of our experiences and the survey, we want to shape the idea of *Research Software Engineering*, not for a single-person-proof-of-concept-prototype-project-at-a-University but for highly distributed collaborative projects like EU projects. We want to find out how tools and workflows have to be adopted for that purpose. A major issue within this concept is the motivation of each contributor because enforcements and strict rules — possibly combined with some kind of coercion — will neither work nor strengthen collaboration.

Consequently, agile tools (e.g. version control, documentation and continuous integration) should be woven into a framework that provides the technical basis for software engineering but does not formulate strict rules. Such an approach would rely on weak processes and the provision of basic tools. Motivation to follow the processes comes from self-commitment and through improving the awareness for the "social responsibility" everyone carries: since everyone participates in the development process, everybody benefits or suffers from the contributions of the others. This way, a community can be formed. Additional mechanisms to explore are how to trigger motivation through market-based models (e.g. [4]) or through sportive competition (e.g. [15]). It could be based on incentives or status within the community. The social value of individual contributions to the project can be valued based on the quality of code or provided documentation (e.g. [9]). However, it is important that the evaluation of the individuals' contributions comes from the community and not from one commanding entity judging the work. Of course, there is no mechanism to punish contributors as this would be a dead end for motivation. We expect from this approach an increase in maintainability and understandability of the software sources, and hence more reuse of code and components.

Future work will cover the introduction of tools tailored to this kind of software development, and the analysis of different aspects of incentives systems for distributed development. This includes observations of effects on the individuals' motivation as well as development of communities and their reactions.

Acknowledgment

The research reported here was supported by the HYDRA EU project (IST-2005-034891). Thanks to Markus Eisenhauer for his hints on the survey design. We are deeply indebted to our survey participants for their faith in us and the time they spent with our questionnaire.

References

[1] IEEE Std. 829 – software test documentation, 1998.
[2] IEEE Std. 1074-2006: Standard for developing a software project life cycle process, 2006.
[3] N. Ayewah, D. Hovemeyer, J. D. Morgenthaler, J. Penix, and W. Pugh. Using static analysis to find bugs. *IEEE Software*, 25:22–29, 2008.
[4] D. F. Bacon, Y. Chen, D. Parkes, and M. Rao. A market-based approach to software evolution. In *24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 973–980, New York, NY, USA, 2009. ACM.
[5] E. Carmel and R. Agarwal. Tactical approaches for alleviating distance in global software development. *IEEE Software*, 18(2):22–29, March/April 2001.
[6] M. Cataldo and S. Nambiar. On the relationship between process maturity and geographic distribution: an empirical analysis of their impact on software quality. In *ESEC/FSE '09*. ACM, 2009.
[7] L. E. Deimel and M. Pozefsky. Implementation of programming standards in a computer science department. In *Proceedings of the 17th Annual Southeast Regional Conference*. ACM Press, 1979.
[8] B. J. Dempsey, D. Weiss, P. Jones, and J. Greenberg. Who is an open source software developer – profiling a community of linux developers. *Communications of the ACM*, 45(2):67–72, February 2002.
[9] S. Dencheva, C. R. Prause, and A. Zimmermann. Collaborative moderation - fostering creativity with a corporate wiki. In *Workshop on Methods & Tools for Computer Supported Collaborative Creativity Process: Linking creativity & informal learning*, 2009.
[10] H. Erdogmus. Déjà vu: The life of software engineering ideas. *IEEE Software*, 27:2–5, 2010.
[11] N. E. Fenton and S. L. Pfleeger. *Software Metrics — A Rigorous and Practical Approach*. Thomas International Computer Press, second edition, 1997.
[12] M. Grechanik. Attracting industry partners to software engineering research. *SIGSOFT Softw. Eng. Notes*, 34(6):4, 2009.
[13] T. Niinimäki and C. Lassenius. Experiences of instant messaging in global software development projects: A multiple case study. In *International Conference on Global Software Engineering (ICGSE'08)*. IEEE Computer Society, 2008.
[14] C. Prause, J. Kuck, S. Apelt, R. Oppermann, and A. B. Cremers. Interconnecting documentation - harnessing the different powers of current documentation tools in software development. In J. Cardoso, J. Cordeiro, and J. Filipe, editors, *Proceedings of the Ninth ICEIS*, volume ISAS, pages 63–68. INSTICC, 2007.
[15] C. R. Prause and S. Apelt. An approach for continuous inspection of source code. In *Proceedings of the Sixth International Workshop on Software quality (WoSQ)*, New York, NY, USA, 2008. ACM.
[16] C. R. Prause, M. Scholten, A. Zimmermann, R. Reiners, and M. Eisenhauer. Managing the iterative requirements process in a multi-national project using an issue tracker. In *3rd International Conference on Global Software Engineering*. IEEE, 2008.
[17] J. Richardson and W. Gwaltney. *Ship it! A Practical Guide to Successful Software Projects*. Pragmatic Bookshelf, 2005.
[18] P. C. Rigby, D. M. German, and M.-A. Storey. Open source software peer review practices: A case study of the apache server. In *International Conference on Software Engineering (ICSE'08)*. ACM, 2008.
[19] N. Rozanski and E. Woods. *Software Systems Architecture*. Addison-Wesley, 2005.
[20] B. Sechser. The marriage of two process worlds. *Software Process Improvement and Practice*, 14:349–354, June 2009.