How can Academic Software Research and Open Source Software Development help each other?

Vamshi Ambati
Institute for Software Research International
Carnegie Mellon University
vamshi@andrew.cmu.edu

S P Kishore
Language Technologies Institute
Carnegie Mellon University
skishore@cs.cmu.edu

Abstract

In this paper we discuss a few issues faced in coordinating, managing and implementing academic software research projects and suggest how some of these issues can be addressed by adopting tools and processes from Open Source Software Development. At the same time we also discuss how a few issues in Open Source Software Development (OSSD) projects can be addressed by adopting processes from Academic Software Research.

1. Introduction

Academic Software Research (ASR) has made substantive contributions in varying degrees for the growth of Software and Technology in diverse areas like data mining, bioinformatics, astronomy, natural language processing, medicine and others. With the wide spread of the Internet, many universities and researchers are working on collaborative projects from geographically distant areas. A typical example is the Universal Digital Library [7] project where universities and researchers from India, China, and the United States are collaborating on massive distributed sub-projects like document acquisition, digitization, quality control etc. However, characteristics of Academic Software Research like rapid prototyping, less documentation, frequent inflow and outflow of project members (e.g., students graduating, researchers changing positions) though useful and productive in collocated projects, are beginning to create problems in the academic community as the projects scale and move into geographically distributed environments. The characteristics of Academic Software Research and the issues mentioned above are essentially deviations from the conventional software engineering practices and it is worth investigating different paradigms of software development and imbibing processes and practices into the software development cycle of academic software research.

In this paper, we take the position that a few problems of communication, coordination and quality that creep into Academic Software Research as they scale and distribute geographically can be solved by adopting practices and tools from OSSD and that issues like entry barrier in the developer community for a new comer and gap between Developer and non-developer community exhibited in some OSSD projects can be addressed by adopting practices from academic software research.

This paper is organized as follows. Section 2 discusses characteristics and issues that crop up in Academic Software Research. In section 3 we discuss Open Source Software development as an emerging paradigm in Software Engineering and the issues faced by OSSD. Section 4 suggests tools and processes that can be adopted by Academic Software Research while section 5 discusses how OSSD can benefit from processes adopted from Academic Software Research. Finally, we conclude by pointing out a necessity for new tools that are needed for the benefit of both the academic and open source communities.

2. Academic Software Research (ASR)

We define Academic Software Research as a team based activity in which students or researchers in Academic institutions involve themselves in building large scale software systems with the guidance of faculty or senior researchers. These projects are mostly funded and usually managed by a single institution including various other departments in it .With the advent and extensive use of internet for software development, the scenario is changing and projects are now involving groups from globally distributed institutions. However, to make development easier these projects are always divided into sub problems and the researchers are allowed to pick the problem that they are interested in. As a result a single person or a small group gets to work on each of the sub problems identified.

Although results are promising in academic software, academicians complain that in massive distributed projects involving various academic institutions issues of collaboration, communication and control creep in which if handled appropriately would enable the successful completion of projects in a cheaper and elegant way. Another issue that affects academic software research adversely is projects in which an active developer opts out of the project suddenly or leaves the institution after the completion of his academics. There is a lot of contextual information and assumptions involved in the project, made by that developer at various moments of time and all these are now lost and the code is the only artifact that remains. This makes the learning period of a new comer longer and sometimes the code becomes so obsolete and un-understandable that the new comer opts to redo the code himself. This introduces an unexpected delay in the project and can also cause unforeseen problems to the entire project if this group is a critical component for the project as a whole. Some of the other issues which are indirectly related to the already mentioned are:

1) Documentation in research projects is very sparse. Research involves results and achieving these results becomes the primary goal during the developmental phase and so documentation is usually done at the end of development. But research projects often don't work on a deadline. And even if they did work that way, it is very unlikely that the person who starts the project is the same as the person who finishes it. As a result, in the due course the documentation part is lost which makes things tougher at the end.
2) In Academic software research the person who works on the module is the person who knows the most about it and the bugs in it. Bug tracking is very specific to the group or developer. Other groups are only concerned about the results of the research and not how it works. As a result, when the project scales these bugs grow and lead the researcher into performing quick fixes which turn out to be hazardous to the quality and the long term goals of the project.

Methods and Tools which address these issues should evolve. A few suggestions are made in the final sections of the paper, but there is a requirement for more of them. We strongly feel that if these issues are addressed completely, academic software research can be more productive and produce a double fold of the existing outputs.

3. Open Source Software Development

The open source model of software development has gained the attention of the business, the practitioners and the research communities [4]. It is characterized as a fundamentally new way to develop software [3].

Open source developments typically have a central person or body that selects some subset of the developed code for the "official" releases and makes them widely available for distribution [1]. Tasks are not assigned; any person interested in a particular aspect of development can choose to participate and contribute to it. There is no explicit system-level design or even detailed design [3] other than in the heads of a few set of core developers. Communities for different phases of software development are formed. Every interested and motivated person joins the community that suits his level of interest and expertise. Usually the set of core developers is very small and they have the editing and commit privileges over the modules in the code. Then there are several other communities each consisting of potentially large numbers of volunteers for bug-fixes, testing, bug-reporting, documentation, release management etc. The communities are globally separated and co-ordination between them takes place in ad hoc ways [1]. Code is written with more care and creativity because developers are working only on things for which they have a real passion.

Though the quality and dependability of today's open source software projects have proved to be roughly on par with commercial developed software, we feel there are several areas where there are opportunities for improvement. For example often new developers have to understand the code and community practices and culture in order to contribute to the project. And there isn't much documentation as most of the OSS projects can not afford extensive documentation. Therefore it remains an obstacle for the initiation and infusion of new members into existing communities. Another significant issue is that the feedbacks of non-developers are not often heard on the mailing lists. This we feel is because the feedbacks from non-developers are treated as unpractical by the developers. Bringing awareness of the project and work culture and creating familiarity with tools among the huge user community seems a challenge. If it can be achieved we would see more users joining the developer community and this would result in improved and constructive interactions between the different communities.

4. OSSD helps ASR

In order to address issues in academic software research mentioned in section 2 we reverted to the two most popular Software Engineering paradigms in the present world – Proprietary and

Open Source Software development, to see if any of them have a closest possible solution.

For years now, Proprietary Software development has successfully implemented methods of enforcing opt-out rules or agreements or by producing extensive documentation to avoid issues that might arise due to persons leaving projects abruptly. Such methods clearly can not be adopted in Academic software research as no one can be bound to agreements in Academia. Also, the communication methods of Proprietary Software development are person to person communications and do not scale as size and complexity quickly overwhelm communication channels [1]. Ad hoc communication is always necessary however, as a default means of overcoming coordination problems [1].

We then turned to Open Source Software Development which bears common similarities with Academic software Research [2]. Observing the characteristics of Academic Software Research and the characteristics of OSSD mentioned in the previous sections we support this similarity and propose that academic software research can benefit the most by adopting methods and practices from OSSD.

Adopt existing tools: OSSD has created for itself a set of software engineering tools with features that fit the characteristics of open source development process [5]. It has addressed similar issues of collaboration and communication that Academic Software Research is facing through these efficient set of tools. A comprehensive list of tools that were used in most successful Open source projects and Research projects in various phases of software development is presented in table below. Also, the first step in adopting OSSD processes and methods would require the adoption of these tools [5].

Table 1 Some common tools used in OSSD

Version control	CVS, Subversion, WinCVS, ViewCVS
Issue tracking	Bugzilla, Gnats,DebBugs,Bonsai
Communication	Mailman, IRC, MajorDomo, Ezmlm
Build systems	Ant, Make,Autoconf
Design and code generation	ArgoUM ,XDoclet, Castor
Testing tools	DejaGnu, Tinderbox, JUnit,Lint, CodeStriker
Collaboration Environments	SourceForge , SourceCast

Peer reviews*:* Open source software development has benefited from "peer reviews". They have been recognized as a widely used and important quality assurance process in OSSD. A peer review is an informal review where someone other than the author, either collocated or distributed, checks the code with purpose of finding defects. It also maintains a healthy and competitive environment amongst the developers. In Academic Software Research a particular group of developers or in most cases just one developer is given the responsibility of a sub-problem and then the solution is expected of him. This would sometimes lead to group specific or person specific results. Introducing a "peer review" in Academic Software Research would result in many people to get familiar with the work done in the sub-problem and would assure quality and maintainability to the code.

Community development: In OSSD there are groups of members called "communities" for various phases of the software development. There is a small set of core developers who have the editing and commit privileges and decides what goes into the code. This is the developer community. There are other large communities for each task of bug-fixing, testing, bug-reporting, documentation, release management etc. This helps the developers to stay concentrated on their work and also enables extensive testing and efficient bug fixing. This may not be feasible in Academic Software Research but a reasonable way of implementation of the concept of a "community" in Academic Software Research could be that every sub-group working on a sub-problem could act as a community to the other sub-group.

Several projects at Carnegie Mellon University (CMU) have benefited from the open source practices and methodologies. RADAR [6] project funded by DARPA is a very good example of one such project. Interactions and communications of different groups take place over the mailing lists which not only enable ad-hoc communication but also act as a log of the interactions. Every leader of a subgroup in RADAR is also a member of one or more groups and acts as a reviewer for that group, ensuring the quality across the sub projects.

5. ASR helps OSSD

In this section let us look at how we can attempt to solve some of the issues of Open Source Software development mentioned in section 3. We feel that a significant portion of the issues mentioned arise due to the fact that as projects scale in size like the Apache and Mozilla, it becomes extremely difficult for a new comer to join any of the non user communities. As a result there is a huge gap between the experts in that community and the new comers. We feel that the way Academic Software Research solves this issue can be helpful to the Open Source Software Development and we also feel that adopting this does not violate the characteristics of Open Source Software Development.

In Academic Software Research whenever a new comer comes into the project he looks for people who can teach him and these are often the senior researchers or faculty in that project. In OSSD there has been activity in the mailing lists which creates an asynchronous mode of learning. But since code developers are mostly involved in other activities most of the questions are left unanswered. So, bringing in the notion of a community of teaching assistants would solve this problem to a considerable extent. Among many motivations for which developers work on OSS projects fame, fun and learning are a few. A good way of learning is by teaching. The members in such a "teaching community" would be people who are in OSSD just for learning and are self-motivated and willing to teach what they have learnt. The result of growth of such a community would make the non-developers more knowledgeable and also creates a way for more users to join the developer community which is good for further progress in the project.

6. Need for new tools

We need more tools to adequately address the issues in both Academic Software Research and Open Source software development. A potential artifact that we have figured out in any software development involving research is an intermediate result. The problem of a researcher or a developer leaving the project can only be addressed in its full if we can completely know and understand these results and the various issues that the developer treaded upon while achieving them. Results indicate the progress of the project. The research community needs tools that keep track of results and the configurations of the project that lead to such results.

Although there is a wide range of tools available in OSSD, in order to support many other software engineering practices like requirements management, project management, metrics estimation, scheduling, program analysis and test suite design etc [5] we still need more of them. The open source community and the Academic Software Research community should involve in building these tools for the benefit of the two communities.

7. Conclusion

We have tried to raise issues in Academic Software Research and suggested that they can be addressed by adopting from OSSD tools for development and collaboration and processes and methodologies like "peer reviews" and community development. We also discussed issues in OSSD and suggested a "teaching community" as a possible solution for some of the issues.

8. Acknowledgements

We are indebted to the valuable suggestions and comments of Dr. Raj Reddy (CMU), Dr. William Scherlis (CMU), Dr. Anthony Tomasic (CMU) and Timothy J Halloran (CMU).

9. References

[1] Mockus, A., Fielding, R., & Herbsleb, J.D. Two Case Studies of Open Source Software Development: Apache and Mozilla (2002). *ACM Transactions on Software Engineering and Methodology, 11, 3,* pp. 309-346.

[2] Nikolai Bezroukov „Open Source Software Development as a Special Type of Academic Software Research" URL:http://www.firstmonday.dk/issues/issue4_10/bezroukov/index.html

[3] P.Vixie, "Software Engineering," in Open Sources: Voices from the Open Source Revolution, C Dibona, S Ockman and M.Stone, Eds. Sebastopol, CA:O'Reilly, 1999, pp.91-100.

[4]. Capiluppi A., Lago P., Morisio M., 2002, "***Characterizing the OSS process***", at the 2nd Workshop on Open Source Software Engineering, Int. Conf. Software Engineering, May 2002

[5] Jason E Robbins (2002) "Adopting OSSE Practices by Adopting OSS Tools". Chapter in Perspectives on Open Source and Free Software,J.Feller, B.Fitzerlad et al.

[6] The RADAR project (CMU) http://www.radar.cs.cmu.edu/

[7]The Universal Digital Library Project http://www.ulib.org/