Out of the Lab, Into the Wild: Growing Open Source Communities around Academic Projects

Academic researchers develop large amounts of software, be it for validating a hypothesis, for illustrating a new approach, or merely as a tool to aid some study. In most cases, a small focussed prototype does the job, and it is disposed quickly after the focus of research moves on. However, once in a while, a novel approach or upcoming technology bears the potential to really change the way in which a problem is solved. Doing so promises professional reputation, commercial success, and the personal gratification of realizing the full potential of a new idea. The researcher who made this discovery then is tempted to go beyond a prototype towards a *product* that is actually used – and is faced by a completely new set of practical problems.

The Fear of the User

Frederick P. Brooks, Jr., in one of his famous essays on software engineering, gives a good picture of the efforts related to maintaining real software, and warns us of the user:

> "The total cost of maintaining a widely used program is typically 40 percent or more of the cost of developing it. Surprisingly, this cost is strongly affected by the number of users. More users find more bugs."[1]

While this figure might well be different in today's environment, the basic observation is still true, and may even have been aggravated by the use of instantaneous global communication. Even worse, more users not only find more actual bugs, but also articulate more wishes in general. Be it a genuine error, a feature request, or merely a fundamental misunderstanding of the software's operation, the typical user request is far from being a technically precise bug report. And each request demands the attention of the developers, consuming precious time that is not available to actually write code.

The analytical mind of the researcher foresees this issue, and, in its natural struggle to prevent a gloomy future in customer care, may develop an outright *fear of the user*. In the worst case, this may lead to a decision against the whole project, in a weaker form it may still lead researchers to practically hide brilliant software products from potential users. More than once have I heard researchers saying: "We don't need more visibility, we are getting enough emails already!" And indeed, there are cases where the communication effort related to a software tool exceeds the effort that a researcher can invest without abandoning her main job.

Often, however, this tragic outcome could easily have been prevented. Brooks could hardly foresee this. When he wrote his essays, users were indeed customers, and software maintenance was part of the product they purchased. A balance had to be found between development effort, market demand, and pricing. This is still the case for many commercial software products today, but has little to do with the reality of small-scale Open Source development. Typical OSS users do not pay for the service they receive. Their attitude accordingly is not that of a demanding customer, but more often that of a grateful and enthusiastic supporter. No small part of the art of successful OSS maintenance is turning this enthusiasm into much needed support, balancing the increase in user interest with an increase in user contribution.

[1] Frederick P. Brooks, Jr.: The Mythical Man-Month. Essays on Software Engineering. Anniversary Edition. Addison-Wesley, 1995.

Recognizing that Open Source users are not just “customers who don’t pay” is an important insight. But it must not lead us to overestimate their potential. The optimistic counterpart of the irrational fear of the user is the belief that active and supportive Open Source communities grow naturally, based merely on the license that was chosen for publishing code. This grave error of judgement is still surprisingly common, and has sealed the doom of many attempts of creating open communities.

Sowing and Reaping

The plural of “user” is not “community.” While the former may grow in numbers, the latter does not grow by itself, or grows wildly without yielding the hoped-for support for the project. The task of the project maintainer who seeks to benefit from the users’ raw energy therefore resembles that of a gardener who needs to prepare a fertile ground, plant and water the seedlings, and possibly prune undesired shoots before being able to reap the fruits. Compared to the rewards the overall effort is little, but it is vital do the right things, at the right time.

Preparing the Technical Ground Building a community starts even before the first user appears. Already the choice of the programming language determines how many people will be able to deploy and debug our code. Objective Caml might be a beautiful language, but using Java instead will increase the amount of potential users and contributors by orders of magnitude. Developers thus must compromise, since the most widespread technology is rarely most efficient or elegant. This can be particularly hard step for researchers who often prefer superiority of language design. When working on Semantic MediaWiki, I have often been asked why in the world we would use PHP when server-side Java would be so much cleaner and more efficient. Comparing the community size of Semantic MediaWiki with similar Java-based efforts may answer this question. This example also illustrates that the target audience determines the best choice of base technology. The developer herself should have the necessary insight to make a most opportunistic decision.

Thoroughly Working the Ground A related issue is the creation of readable and well documented code from the very start. In an academic environment, some software projects are touched by many temporary contributors. Changing staff and student projects may deteriorate code quality. I remember a small software project at TU Dresden that had been maintained quite well by a student assistant. After he had left it was found that his code was thoroughly documented – in Turkish. A researcher can only be a part-time programmer, so special discipline is needed to enforce the extra work needed for accessible code. The reward will be a much greater chance of informed bug reports, useful patches, or even external developers later on.

Spreading the Seeds of Communities Inexperienced Open Source developers often think it as a big step to publish their code openly. In reality nobody else will notice. To attract users and contributors alike one has to spread the word. The public communication of a real project should at least involve announcements for every new release. Mailing lists are probably the best channels for this. Some social skill is needed to find the balance between annoying spam and shy understatement. Projects that are motivated by the honest conviction that they will help users to solve real problems should be easy to advertise respectably. Users will quickly notice the difference between shameless advertising and useful information. Obviously, active announcements should wait until the project is ready. This does not only include its actual code but also its homepage and basic usage documentation.

Throughout its lifetime, the project should be mentioned in all *appropriate* places, including Web sites (start with your homepage!), presentations, scientific papers, online discussions. One cannot appreciate enough the power of the single link that leads a later main contributor to his first visit of the project's homepage. Researchers should not forget to also publicise their software outside of their immediate academic community. Other researchers are rarely the best basis for an active community.

Providing Spaces to Grow Trivially easy, yet often neglected is the duty of project maintainers to provide for the communication spaces that communities can grow in. If a project has no dedicated mailing list, then all support requests will be sent privately to the maintainer. If there is no public bug tracker, bug reports will be fewer and less helpful. Without a world-editable wiki for user documentation, the developer is left with extending and refining the documentation continuously. If the development trunk of the source code is not accessible, then users will not be able to check the latest version before complaining about bugs. If the code repository is inherently closed, then it is impossible to admit external contributors. All of this infrastructure is offered for free by a number of service providers. Not all forms of interaction might be desired, e.g. there are reasons to keep the group of developers closed. But it would be foolish to expect support from a community without even preparing the basic spaces for this.

Encouraging and Controlling Growth Inexperienced developers often are concerned that opening up mailing lists, forums, and wikis for users will require a lot of additional maintenance. It rarely does, but some basic activities are of course necessary. It starts with *rigorously enforcing* the use of public communication. Users need to be educated to ask questions publicly, to look up the documentation before asking, and to report bugs in the tracker instead of via email. I tend to reject all private support requests, or to forward the answers to public lists. This also ensures that solutions are available on the Web for future users to find. In any case, users should be thanked explicitly for all forms of contribution – many enthusiastic and well-meaning people are needed for building a healthy community.

When a certain density of users is reached, support starts to happen from user to user. This is always a magical moment for a project, and a sure sign that it is on a good path. Ideally, the core maintainers should still provide support for tricky questions, but at some point certain users will take the lead in discussions, and it is important to thank them (personally) and to involve them further in the project. Conversely, unhealthy developments must be stopped where possible, and in particular aggressive behaviour can be a real danger to community development. Likewise, not all well-meant enthusiasm is productive, and it is often necessary to say no, friendly but clearly, to prevent feature creep.

The Future is Open

Building an initial community around a project is an important part of transforming a research prototype into a grown Open Source software. If it succeeds, there are many options for further developing the project, depending on the goals of the project maintainer and community. Some general directions include:

- Continuing to grow and develop the project and its OSS community, enlarging the core team of developers and maintainers, and eventually making it independent of its academic origin. This may involve further community activities (e.g. dedicated events) and maybe establishing organizational support.
- Founding a company for commercially exploiting the project based on, e.g., a dual-license or consulting business model. Established tools and vibrant communities are a major asset for a start-up company, and can be beneficial to several business strategies without abandoning the original OSS product.
- Withdrawing from the project. There are many reasons why one may no longer be able to maintain the close affiliation to the project. Having established a healthy open community maximizes the chances that the project can continue independently. In any case, it is much more respectable to make a clear cut than to abandon the project silently, killing it by inactivity until its reach is diminished to the point where no future maintainer can be found.

The shape of the community will be different when working toward one of these principal options. But in each case, the role of the researcher changes in the cause of the project. The initial scientist and coder may turn into a manager or technical director. In this sense, the main difference between an influential OSS project and a perpetual research prototype is not so much the amount of work but the type of work that is required to succeed. Understanding this is part of the success – the only other thing that is needed is an awesome piece of software.