Open Source Development: An Arthurian Legend

Jonathan E. Cook
Department of Computer Science
New Mexico State University
Las Cruces, NM 88003 USA
jcook@cs.nmsu.edu

ABSTRACT

OSSD (Open Source Software Development) achieves remarkable success in delivering complex software systems – systems which are incredibly reliable and robust – in a short amount of time and without even paying anyone! Naturally, in the face of this success, organizations are interested in seeing if the mechanisms behind OSSD success can be migrated into their own practices, hopefully improving their systems and their productivity.

In this paper, we look (lighthearted at first) at the motivations behind those involved in OSSD and describe the problems that need to be overcome if OSSD-type practices can be migrated into traditional organizations.

1 IN THE BEGINNING

Once upon a time in a land far, far away, there lived an average boy in an average village, whose name was Arthur. At least he thought he was average, but as it turns out he was born of the noblest blood in the land, and destined for greatness. As he grew, Arthur was filled with a sense of duty, honor, and of the possibility of achievement. He did not believe that adventures were too big or too hard or too dangerous for mere individuals to undertake, rather he was filled with the ideas of opportunity.

Arthur grew strong and intelligent, and applied himself diligently to his studies. He found he had a great fascination for the tools known as "computers", and in particular the software that drove them. Early on he talked his parents into buying him an Agima, a popular and inexpensive computer. He dissected it inside and out, learned how it worked, learned the operating system and learned how to program it.

This work was supported in part by the National Science Foundation under grant CCR-9804067 and the Department of Education under grant P200A70303-97. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

When he went to the university, he found a world of other computers and endless possibilities. Yet this world was stifled by the lack of available software. Oh, sometimes this software was available to those who practiced the dark arts of business and had the cash available to purchase it, but the potential of the computer was hampered by the lack of freedom of all people to pursue using the computer in their lives.

Arthur made a fateful decision one day when a particular application was unavailable to him. History has lost what particular application this was, but it is unimportant whether it was a word processor, an operating system, a graphics environment, or a presentation editor. Arthur decided that *his* adventure was to write this application himself and give it to the world. Yes, the naysayers told him that such an application took hundreds of skilled laborers many months to complete, but he dismissed their gloom and doom and set off anyways.

After devoting several months in a monk-like state of undertaking his adventure, he had completed a first version of his application. Sure, it didn't have the bells and whistles of the type that the dark arts produced, but it was functional nonetheless. He boldly stepped into the public square and announced his application to the world, freely allowing anyone to copy it and even see and use its source code. He called his system Camelot.

Friends and foe alike were astonished at Camelot, and the fact that Arthur alone had built it. He was acclaimed as a hero of the people, and in the eyes of the people he was king of Camelot and a standard bearer of honor and freedom.

A curious thing happened after he released Camelot. Other adventurers of like mind began adding on to his system, building all of those bells and whistles and advanced features that the dark systems included. Such was the magnitude of the effort that Camelot soon surpassed the dark systems in functionality and even reliability. The most capable of the adventurers were invited by Arthur to join him at the Round Table, a place from which he would direct the future of Camelot, but only

to the benefit of the people and of freedom.

The Knights of the Round Table achieved hero status among the people, and many young adventurers-to-be aspired to become a knight, or even another Arthur himself. They would tirelessly practice, submit code to be reviewed by the knights and possibly included into Camelot itself, for to be named as a contributor to Camelot was a great honor. Other public and open software systems were created, and they had similar success. Each of these became a benevolent kingdom in its own right, and adventurers flocked to support them, all in the name of honor and freedom.

The success and rise of these kingdoms caused great concern among the purveyors of the dark arts. Their hold on the people was diminishing, and their future looked dim. It seemed that their own laborers, skilled as they may be, produced far less than the free adventurers. They began to wonder if whether or not this revolution would be their demise...

2 OSSD MOTIVATORS

Although the introduction was light-hearted, it does indeed elucidate some of the aspects of open-source software development (OSSD) that make it succeed so well. In this section, we step through individual aspects and explain how they fit in or conflict with the structure of a typical closed-source software development (CSSD) organization.

The Benevolent Dictatorship

Perhaps in a balance with its product goal of freedom, the *process* of OSSD tends to be quite authoritarian. The founder, or current maintainer, of the system normally has complete control of the changes and additions to the system, with essentially no recourse for those who attempt to contribute to the system. This is accepted because everyone assumes, with good reason, that the person in control acts for the benefit of all, not just themselves. When a project becomes big enough, this person often works in close conjunction with a core team of advisors.

The analogous role in CSSD is the project manager and perhaps a few select technical leads as a core team. They may have strong authority over the project, even complete in relation to the project team, but the history of business usually shows us that influences from further up the chain of command can often have major and possibly disastrous effects on the project. It is essentially the old cliche of "marketing rather than technical decisions". When those involved in the project see the authority but not the benevolence, any hope of OSSD-like success is dimmed considerably. In the worst case the project becomes a "death march" [4].

The Pool of Resources

Resources being people, machines and time, OSSD is unconcerned with the amount of resources that may be available, or even with the amount of resources that are used in the construction and maintenance of a project. Since these resources are contributed willingly, there is no desire or need to measure, track, or control them. CSSD, on the other hand, essentially *must* keep the amount of resource usage below that of the expected income from the product. To be able to do this, it must measure and track resource usage.

An effect of this is that although it may appear that OSSD achieves remarkable goals with little resources usage, in fact we do not really know. We have no idea how many would-be contributors tried to develop some source and failed, or how many stopped when they saw someone else complete the same functionality first. We might be able to obtain data about how many contributions were submitted and how many out of those were actually incorporated into the source, but that does not necessarily give an estimate of the true OSSD cost.

In other words, CSSD might not be so bad after all.

The Level of Talent

It is no secret that the biggest single factor in programmer productivity is the individual. Well-established results point to at least two orders of magnitude difference in productivity, and some studies have shown that perhaps one out of five programmers actually produces *negative* work – that is, the other four would be more productive without that person.

Certainly much of OSSD success is dependent on this fact. There is little doubt that those who undertake and succeed on the initial construction of an OSS project are at the top end of the talent spectrum. Furthermore, OSSD naturally selects for its contributors the cream of the talent. It does not have to do this with hire and fire decisions – lesser individuals can submit potential contributions, but only the best contributions will actually be incorporated.

CSSD, unfortunately (or not, depending on your viewpoint), cannot simply rely on the cream for its resources. In plain English, we cannot all rise to be Arthur, or even a knight of the Round Table. It has always been a delicate balance just to get enough individuals interested in pursuing a technical career, much less eliminating most of them to concentrate on the "cream". The plain fact is that most CSSD projects will be staffed with a team that reflects the normal distribution of technical talent. Often there is at least one person who can be a "hero" and lead the project through to success. But most of the team will be of average productivity.

Hoping to somehow magically reach the success levels of

OSSD[1] when using a different talent pool is akin to believing in Merlin, or (in traditional software engineering mythology) a silver bullet.

Motivations for Contribution
Of course, we all know that people can be motivated to do great things. Rarely, if ever, is that greatness achieved through motivations like *money*. OSSD in many ways works through perhaps the two strongest motivations: self-interest and philanthropy. Self-interest exists in most OSSD projects because the initiators want to use the system themselves. They benefit personally. Philanthropy exists because they believe that the (computing) world will be a better place because this certain application is free to other users.

In contrast, CSSD organizations typically pay their programmers to develop software that the organization will sell or otherwise make money off of. There is nothing wrong with this, but this alone is not usually a sufficient motivation for greatness (either in product quality or process efficiency). Many organizations do work on other motivational factors (team spirit, worker accommodations, self-excellence, etc.), and there certainly are and have been products which captured the imagination of a team and drove it to greatness, but this is not the rule, it is the exception.

Moreover, like it or not, there are many, many CSSD projects that, simply put, are fairly boring. The countless number of specialized business applications that come out of in-house IT departments is an easy example. But this software is needed, nonetheless.

Not Redesigning the Wheel
Although one can certainly find exceptions, most OSSD projects are not "cutting-edge" in the sense of embarking on something no one has ever tried before. Indeed, their greatest successes usually come in well-understood domains where the software design is essentially a given. Software engineering has long understood the primacy of getting the requirements and the design right, these being the overwhelming factors for product success. OSSD, in essence, does not even have to worry about requirements, and for the most part gets to choose from the best features of existing designs. We should not be surprised, then, at the speed and quality that the product is developed at.

CSSD, on the other hand, is stuck with the hard job of gathering requirements and making sense of them, attempting unproven designs, and paying the costs of making mistakes in these phases. Nothing in OSSD can offer support for these problems.

3 OSSD POSSIBILITIES

[1]Keep in mind the caveat that we do not know the true cost of OSSD.

So far, this position paper has been pretty gloomy in regards to applying OSSD techniques to improve CSSD organizations. It is true that the author's opinion is that most of what OSSD does is indeed inapplicable to CSSD. However, there are some possibilities for success. Other insights into OSSD have yielded speculation as well (e.g., [3]). Others also echo the sentiment that more needs to be understood about OSSD [1, 2].

Customer Interaction in Certain Domains
In some domains, customers will be or will have access to technical people who are capable of understanding a software system and contributing to it. Whether it is a library or framework that is sold to application developers, an operating system and development environment used by others, or embedded software used by system builders, when these types of users find bugs or need additional capability, they may indeed be able to make the change themselves. An open source model for the product would allow these customers to quickly solve their own problem, and contribute to the product as well.

In this scenario, the self-interest motivator in OSSD is satisfied because the user will probably get the fix much sooner than in a normal CSSD report-bug-and-wait-for-fix type process. The philanthropy motivator does not come into play because presumably they bought the system in the first place. But further monetary rewards could be given for fixes that are incorporated back into the selling product.

Rewarding Talent
A talk with almost any good engineer or software developer will quickly devolve into how their effort is not appreciated or rewarded within the organization. In OSSD, there is no "higher-up" who rewards the effort of the project initiator, since they are the authority, but below that, the rewards are obvious: recognition by the incorporation of one's code (and attribution of such), and achievement of "knighthood" by becoming a core team member of some OSS project. This points to the fact that money is usually not a viable reward, at least in and of itself. Basically, those who achieve simply want the recognition.

Of course, handing out a gold medal to the winner by necessity defines everyone else as the losers. And in an organization, usually this kind of separation is not conducive to teamwork and cohesiveness. But not rewarding achievement may be worse – an encouragement towards mediocrity.

Outsourcing and Contracting
Perhaps the greatest possibility for CSSD organizations is the possibility of employing OSS-like structure to enable "outsourced" contributions. OSSD projects are worldwide and distributed, generally without the con-

tributors meeting face to face. With the Internet continually expanding, the idea of a virtual organization can be taken even further to the point of "free contributors to an organization" – that is, free in the sense of open, not necessarily monetarily free.

Imagine that a CSSD organization allows individuals from around the world to register as a potential contributor, and gives them access to the system, the current project goals and needs, all in an OSS-type of style. Then, contributors can send in contributions, and *if* that contribution is incorporated into the system, that contributor is paid some amount. How to determine that amount is an open problem.

Nevertheless, this model fits the futuristic visions of the Internet, where people work at home or wherever in self-defined jobs. It also rewards talent, in the sense that someone who is good will get many contributions accepted, and will presumably be rewarded proportionally. They might even get an "arthurian" reputation, and begin to be asked to contribute to certain projects, or to direct a project as the technical leader.

Not only does OSSD offer a model for individual contributions from developers spread over the world, it also offers the beginnings of a process definition for managing a project inside this paradigm. This process is centered around the authority of the single leader, or of the core team that manages the project. The process also requires well-defined contribution submission standards, in coding style, coding quality, and testing quality. Usually, new test cases must be submitted with contributions, and contributions can be rejected simply because of lack of testing.

The overall process, thus, is centered around dispersing the effort to the contributors, and leaving the core team to simply manage the selection and integration of contributions. Those that they select have (hopefully) been rigorously tested already, so that the core team needs to be concerned about contribution interaction but not about individual contribution quality (once it has been selected).

4 CONCLUSION

This position paper certainly has not brought out all of the issues and facets of OSSD, but has focussed on the motivating factors for those who involve themselves in OSSD, and how those factors match or diverge from the CSSD environment.

In summary, it is this author's opinion that most of the success of open-source software development (OSSD), if indeed that success is really there, is due to factors that ultimately cannot be transferred to closed-source software development (CSSD) in the business organization.

One of the largest positive lights is that OSSD demonstrates how to organize and manage a wide-spread, loosely-coupled development effort. This alone has implications to CSSD organizations that are interested in pursuing a "virtual organization" or even more loosely-coupled paradigm of project development. However, OSSD allows for the selecting of the best contributions and the "waste of resources" that occurs when all other contributions are essentially thrown away. Whether or not this is fundamental to the success of OSS projects and whether or not some form of this paradigm could be used in a CSS project remains to be determined.

Finally, it is clear that more quantitative studies that push further than [3] are needed in order to understand the true costs of OSSD. The cited study provides good quantitative data, but it will be necessary to evaluate the *cost* of the overall OSSD effort, in order to fully understand how it could be migrated into organizations that are indeed concerned about the bottom line.

REFERENCES

[1] T. Bollinger, R. Nelson, K. Self, and S. Turnbull. Open-Source Methods: Peering Through the Clutter. *IEEE Software*, 16(4):8–10, 1999.

[2] S. McConnell. Open-Source Methodology: Ready for Prime Time? *IEEE Software*, 16(4):6–8, 1999.

[3] A. Mockus, R. Fielding, and J. Herbsleb. A Case Study of Open Source Software: the Apache Server. In *Proceedings of the 22nd International Conference on Software Engineering*, pages 263–272, June 200.

[4] E. Yourdon. *Death March : The Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects*. Prentice-Hall, 1997.