The Development of the OpenACS Community*

Neophytos Demetriou, Stefan Koch, Gustaf Neumann
Vienna University of Economics and Business Administration
Vienna, Austria

Abstract

OpenACS is a high level community framework designed for developing collaborative Internet sites. It started from a university project at MIT, got momentum from the ArsDigita Foundation, and split up into a commercial and an open source version. OpenACS has proven its durability and utility by surviving the death of its parent company (ArsDigita) to grow into a vibrant grassroots collection of independent consultants and small companies implementing diverse and complex Web solutions around the globe for NPOs, philanthropy, and profit. A heritage from this history is a still dominant position of contributors with commercial interests, which in its intensity is above the norm found in open source projects. In this paper, OpenACS with its community is presented as a case study documenting the forces between commercial interests, securing investments, and technical development in a large open source project with a large proportion of commercial involvement.

1 Introduction

The free and open source software world has spawned several projects in different application domains like most notably the operating system Linux together with the suite of GNU utilities, the office suites GNOME and KDE, Apache, sendmail, bind, and several programming languages that have achieved huge success in their respective markets.

In the last years, also the commercial interest in open source software has increased dramatically [34]. This has also lead to changes in many projects, which now include contributors who get paid for their contributions and others, who receive no direct payment. This is also reflected in several recent surveys: For example, Lakhani and Wolf [26] found that 13% of respondents

*A modified version of the paper will be publised as a book chapter in: Miltiadis Lytra, Ambjorn Naeve (eds): Open Source for Knowledge and Learning Management: Strategies Beyond Tools, Idea Group Publishing, Hershey, PA, 2006.

received direct payments, and 38% spent work hours on open source development with their supervisor being aware of the fact (684 respondents from 287 distinct projects). Ghosh [11] reports a group of 31.4% motivated by monetary or career (mostly for signaling competence) concerns in a sample of 2,280 responses. Hars and Ou [18] found a share of 16% being directly paid, Hertel et al. [21] report 20% of contributors receiving a salary for this work on a regular basis with an additional 23% at least sometimes in a survey of Linux kernel developers. Given these results, many projects currently will be composed of a mixture of paid and volunteer participants, thus distinctly deviating from the 'classical' open source development as described most notably by Raymond [38]. This mixture, and the resulting conflicts of interests could have severe results on working styles, processes and also products of open source projects. In this paper, we will present the case study of OpenACS [29], a project between commercial interests, securing investments, and technical development. The OpenACS changed its status several times during its lifetime, starting from a university project from MIT, getting momentum from the ArsDigita Foundation, and lastly splitting up into a commercial and an open source version, where the commercial version failed but the community continues to develop the open source version. While the literature yields discussions and examples on commercial projects going open source or enterprises investing in open source projects [3, 20, 19], most notably the Mozilla project [17], the history of OpenACS seems unique in its complexity. Nevertheless, we propose that this form of biography will increasingly show up in software development projects, and that the repercussions on processes and products are important to analyze.

The structure of this paper is as follows: in the following section, we will describe the history of OpenACS. The turbulent past of the project shaped the management framework with its idiosyncrasies. As a next step we will analyze up to which point the influences of the project management structure and forces of the different stakeholders can be observed from some empirical data obtained from mining the source code management system [16, 40]. The paper will finish with some conclusions.

2 History of OpenACS

2.1 The Early Days

The roots of OpenACS trace back to Philip Greenspun and his work on the site `photo.net` starting in 1995, and "Philip and Alex's Guide to Web Publishing" in 1998 [15]. The project started out as a rapid prototyping framework for web-applications, based on the experiences of `photo.net`. The core elements of the framework were a highly scalable web-server (AOLserver [31]) with a tight integration with relational databases (especially with Oracle and PostgreSQL), and the scripting language Tcl [35].

AOLserver was originally developed by NaviSoft under the name NaviServer, but changed names when AOL bought the company in 1995. AOL

uses this server to run its busiest sites, such as *digitalcity.com*[1] and *aol.com*. It has been reported that as early as mid-1999 multiple AOLserver instances were serving more than 28,000 requests per second for America Online [14]. AOLserver is a multi-threaded application server with well-abstracted database access and connection pooling mechanisms. Scalability is achieved due to the fact that most performance critical functionalities are implemented in C.

AOLserver uses the built-in scripting language Tcl as an extension language [36] to provide a flexible means of composing systems from predefined components. This way, application specific hot spots can be quickly programmed and the server can be extended on the fly (without restarting). Based on this combination of scalability and rapid application development it became possible to develop complex web application in short time.

2.2 ArsDigita

We want to focus now on the historical development of the framework, which deeply influenced the structure of the project and the active development community. In 1997, Philip Greenspun and a group of students mostly from MIT joined forces to produce the world's best toolkit for building scalable community-oriented Web applications. The newly founded company ArsDigita ("Digital Arts") was quickly able to attract high profile clients, such as Deutsche Bank, WGBH (radio and television channel), the World Bank, and Siemens. In early 2000, ArsDigita took $35 million from two venture capital firms (Greylock, and General Atlantic Partners).

With one of the world's largest corporations on their client list, ArsDigita was already defying the conventional wisdom by actively supporting an open source version of its toolkit. The founders of the ArsDigita Corporation also created a nonprofit organization, the ArsDigita Foundation, which sponsored a yearly programming contest for high school students and a free brick and mortar school teaching an intensive one-year course in undergraduate computer science (in 2000).

The underlying engineering was supported by millions of dollars of venture capital spent on hiring PhDs in Computer Science from MIT, CalTech and other major universities across the Atlantic and Europe. At this time ArsDigita employed more than 240 people (see Figure 1), mostly developers, working on the foundation of what is now called OpenACS (more about this later). However, with the advent of the investors the interests shifted away from community supported open source version towards the company's needs. The company decided to redevelop the framework based on Java in order to develop a proprietary enterprise collaboration software. The commercial version was called ACS-Java, while the community supported Tcl-based version was developed further under the name of OpenACS. The proprietary product was never launched and, in 2002, ArsDigita was acquired by Red Hat.

[1] Now, `cityguide.aol.com`.

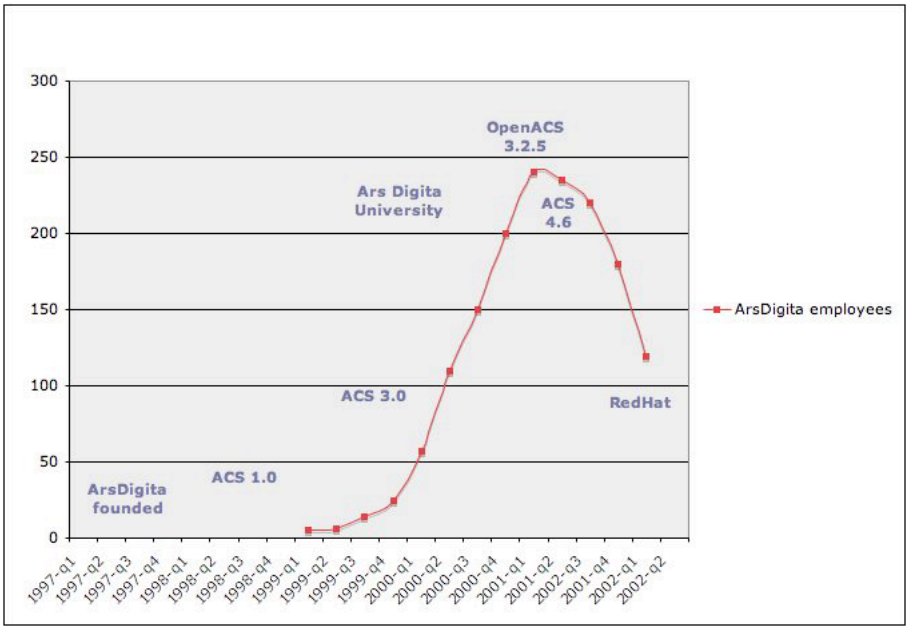

Figure 1: Development of ArsDigita and early OpenACS.

2.3 The ArsDigita Community System (ACS)

In general most Web applications have the same basic needs: (a) user management (represent people and relationships between them), (b) security management (manage accessibility to functionality and data), (c) content management (content repository), and (d) process management.

Instead of developing these functionalities repetitiously on a per-application basis, ArsDigita decided to develop a general application framework addressing these needs. This framework is based on a complex data model (current OpenACS uses a few hundred tables and views) and a high level API in Tcl (in current OpenACS a few thousand functions). This framework was called ArsDigita Community System (ACS) and was first published in version 1.0 in Dec. 1998.

The code base of ACS emphasizes collaboration and management of geographically distributed on-line communities. During the various projects, more and more pieces of code were factored out to increase reuse and to address the requests derived from the collaboration, e-commerce, and content management packages developed on top of the core. The system started with a small core of functionality, but only a little more than one year after the initial version, ACS 3.0 was released as a rich application framework as open source under the GNU General Public License version 2 (Jan. 2000). Dozens of packages based on the core functionality were created. Ten months later, the next major refactoring led to version 4.0 (release Nov. 2000), where a completely

new kernel was developed based on a highly flexible security management system, subsites, templating, workflow management and a refactored content repository. Most notably, the package manager was developed that supports creation of packages with version dependencies, data migration scripts and allowing remote upgrading of packages.

In 2001, the ACS code tree forked inside ArsDigita, with the Tcl code base being maintained and refactored by one group of developers, while the product line was being re-written in Java EE. By 2002, when Red Hat acquired ArsDigita, the Tcl code base became solely supported by the open source community. At this time a rich set of application packages (such as forums, FAQ, bulk-mail, file-storage, calendar, web-shop, etc.) were already available. This rich set of packages led to the adoption of ACS by programmers and companies worldwide and fueled the ongoing development of OpenACS without the strong former commercial backing.

2.4 OpenACS

Since 2002 the OpenACS project is run by a group of independent programmers, whose original goal was to add support for the open source PostgreSQL database to ACS (supporting only Oracle). Soon it was clear that building demanding applications will need more than only the free database, so the community started to fix problems in the code inherited by ArsDigita, porting 3.x packages to version 4.0, writing new ones, improving and extending the core platform, and taking ACS in new directions.

The first important enhancements were two new abstractions, namely the XQL Query Dispatcher for database independence and the Service Contracts API to facilitate greater code reuse, application integration, and package extensibility within OpenACS. ACS 3.x and earlier contained pages with embedded SQL. ACS 4.0 provided named queries and provided APIs onto them. The new XQL Query Dispatcher evolved this idea even further by abstracting these APIs from the application code. The approach involved extracting queries out of the code, and then eliminating the application's need to know about specific implementations of the SQL standard. This proved to be very useful during the porting phase as new RDBMS support could be added on a per-package basis without editing existing code.

ACS 4.0 was based on a thin object system implemented in the relational structure in the database, but allowing a veneer of object orientedness by providing globally unique object IDs, object meta-data, and bundling of data and methods as an object. While this permitted a level of reuse on an object or package basis, it required hard-coding the unit of reuse. Inspired by WSDL, the interface specification for Web services, the Service Contracts API allowed these objects and packages to also create contracts which define their functional level of reuse and customization as well as register their implementation of interfaces, in this way, bringing the level of reuse at the contract level.

The current version is OpenACS 5.2, where numerous user interface improvements, external authentication, automated testing, improved develop-

ment environment, etc. where introduced over the years. And, many application packages were added.

2.5 .LRN

As the most general requirements were quite well supported, more specific needs became addressed by the community. Two important sub-projects started to emerge, namely .LRN [32] and ProjectOpen [30] (an ERP-system based on OpenACS, with currently 4,500 installations [5]). We address in this paper only the case of .LRN.

In 2002, MIT's Sloan School of Management contracted the company OpenForce to develop .LRN to replace their aging Course Management solution (SloanSpace, built on ACS 3.x [27, 13]). The primary goal was to address MIT Sloan's specific needs, but the project had a broader vision than internal deployment.

This investment from MIT provided a strong impulse for the community after such a tumultuous period for the OpenACS project. On the one hand, .LRN became a full-featured course management system with rich community support, providing a substantial promotion of OpenACS as a platform. On the other hand, there was a big part of the OpenACS community comprised of volunteers and developers working on projects, large and small, that had nothing to do with educational technology. These developers preferred OpenACS like it was, basing development decisions on general technical needs, and not on the requirements of the founding project. At the same time, MIT Sloan School wanted to secure their investments, such that no complete split-off from OpenACS is needed, and that general future versions keep functional with future versions of OpenACS, so that .LRN can benefit from future enhancements of OpenACS.

When .LRN 1.0 was released in 2002, some learning management system (LMS) vendors had already started to disappear from the market. Learning institutions feared that a relationship with one vendor would not prove to be of a long-term nature because of the vendors' inability to stay in the market. Different universities had already started to develop learning management based on OpenACS (e.g. Vienna University of Economics and Business Administration [1] or UNED in Spain). The perspectives of developing a common community supported learning management system based on OpenACS was very appealing.

However, the conflicting goals led to an inevitable governance plan discussion with lead institutions seeking formalized management structures to secure the investments of the funding organizations. The .LRN Consortium was founded, which is a non-profit organization with a clearly defined governance and membership structure. The consortium is guided by a board of directors and sees its mission in "creating and supporting a freely available suite of web based educational applications to support learning communities". This can be seen as a form of 'guarding the commons' [33]. Furthermore, OpenACS and

.LRN became more attractive to new people, mainly consultants and organizations. But still, some feel that it changed the community, as these new people are not interacting with the community the same way the old grassroot hackers did.

Today, .LRN 2.x is the world's most widely adopted enterprise-class open source software for supporting e-learning and digital communities [32] and it is installed at prestigious universities like Sloan School of Management at the MIT, the JFK School of Government at Harvard University, or the Universities of Mannheim and Heidelberg in Germany, the Vienna University of economics and Business Administration in Vienna, or the University of Valencia and the Open University of Spain (UNED, Universidad de Educacin a la Distancia with about 200.000 students).

3 The OpenACS Project Management Framework

When OpenACS started as a group on SourceForge to create the ACS port to PostgreSQL, the lead developer gave write permissions to anyone who showed enough competence to help out. The group soon grew to more than 20 people, with about 5 active developers. Organization, collaboration, and feedback helped produce a quality product but, in recent years, maintaining a stable, releasable, progressive codebase has become quite a difficult task. We will address these reasons in the following paragraphs.

Designing and evolving these structures has been an important aspect of open source software development for some time. While preconceptions might think of such projects as completely anarchic, this is most often not the case in reality. An important point to consider is the balance between anarchy and control, as Holck and Jorgensen [22] describe in their account of the organization and process models in the FreeBSD and Mozilla project. They describe the technological infrastructure, but more importantly, the work organization which includes top-level management, module owners, reviewers and committers and the process models for releases (the FreeBSD release process is also described in more detail in [23]) and contributions. In all aspects, there is an approach to strive for a balance between openness towards new participants and contributions, and the need for control, with the acknowledgment that this balance might shift over time. This point is also emphasized up by the paper of Ye et al. [44], which stresses the co-evolution between the system and the underlying community. Using a set of case studies, they define three types of projects (exploration-oriented, utility-oriented and service-oriented) and evolution patterns between those types. Also Erenkrantz [8] in his account of different release management structures in open source projects stresses the point of decentralization and controlling authority as important factors, as does Gallivan [10] under the aspects of trust and control.

3.1 Source Code Management

OpenACS development is maintained in a central source code repository based on the Concurrent Versions System (CVS[2]) [6, 9, 37]. Only developers with write privileges, so called *commiters*, are allowed to make changes to the repository. A developer without these privileges will have to go through a commiter in order to get contributions added to the repository in form of a patch:

> *To contribute a small fix, if you do not have a developer account, submit a patch.*

The code is divided into packages and for each package, one person is designated as the package owner or maintainer. In the past, the package owner was the only one who had the authority to check in changes or elect other programmers to do so. Low responsibility for some packages led the OpenACS team to revise the CVS guidelines:

> *If you are making many changes, or would like to become a direct contributor, send mail to the Core Team asking for commit rights. You can then commit code directly to the repository.*

Technically, everyone with CVS commit rights can commit changes to the code base. This is sometimes required, since OpenACS has now more than 200 packages and not all package owner are always available, but at the same time, this lowers the perceived responsibility of a package owner. According to the CVS guidelines, another way to contribute code to the OpenACS is to add a new package.

> *Contact the Core Team to get approval and to get a module alias.*

The analysis of the public CVS repository shows that over a hundred different people had CVS commit rights since the establishment of the repository in Spring 2001. Rather than requiring developers to coordinate with each other to synchronize efforts, CVS enables developers to update repository files continually and in parallel. Today, OpenACS is a complex system despite the seeming simplicity of its components. A system based on uncontrolled changes to the source code repository is no longer appropriate for a system of this complexity as it greatly inhibits integration, release, and regression testing.

The current OpenACS development team is more diverse than the original team; they live in different time zones, speak different languages, have different needs and requirements, and different coding style. A purely CVS based codebase does not serve the product well in this environment.

3.2 Technical Improvement Proposals (TIPs)

On openacs.org, nearly 10,000 users are currently registered. The website list currently more than 120 community-sites based on OpenACS (including Greenpeace Planet, United Nations Industrial Development Organization

[2]The repository is reachable via the Internet via `http://cvs.openacs.org/` and `http://eye.openacs.org/`.

(UNIDO), Creative Commons, or AISEC (the world largest student organization) and lists 58 companies providing commercial support for OpenACS. The largest sites have up to 2 million users [39]. Since many of the OpenACS contributors are consultants, often in charge of running sometimes dozens of sites, code changes that introduce incompatibilities are very expensive.

Therefore the community adopted a guideline [2] about dealing with changes in the CVS. This guideline requires a so called Technical Improvement Proposal when an update involves any changes the core data model, or will change the behavior of any core package in a way that affects existing code (typically, by changing public API), or is a non-backwards-compatible change to any core or standard package. The first version of the Core Team governance document was released in May 2003, and serves since then as an instrument to guard against changes in the core product. Since 2003 there were between 18 and 25 TIPs accepted per year, about 25% of the TIPs are rejected.[3] While the TIPs provide an instrument to secure investments, they effectively require a sideway development based on coexistence: while changing existing APIs require a TIP, the development of new functionality does not. Currently it appears that the only way to make complex architectural changes is to build a coexisting sub-framework which has certainly disadvantages from the software engineering point of view. In any large software systems, evolution tends to create increasing complexity [4], a fact also acknowledged in studies on open source systems [41], necessitating architectural repair actions, as described by Tran et al. [42] in the context of the Linux kernel and the VIM text editor.

3.3 Bug Tracking and Fixes

A common project risk is to remain unaware of the existence of a major problem beyond the stage at which it can be contained and corrected. OpenACS addresses this problem by the bug-tracker, a software tool for tracking bugs and feature requests. The bug tracking tool was developed using OpenACS and incorporates ideas from BugZilla, Bughost.com, and FogBUGZ.

Figure 2 shows the underlying workflow of the bug management in OpenACS. A bug can be in the state *Open*, *Resolved* or *Closed* and is assigned priority and severity. In a true open source fashion ("given enough eyeballs, all bugs are shallow" [38]) everyone can report bugs and everyone is encouraged to do this. Also Villa [43] describes a bug tracking system based on Bugzilla in a large open source project, GNOME, and highlights the importance of being open while applying the necessary triage to control to possibly massive amount of bug reports.

The OpenACS bug-tracker assigns a bug per default to the package owner. At this state the bug tracker supports an open discussion of the problem in a weblog style. Any user can submit candidate patches for this bug. The pack-

[3] The number of TIPs per year are decreasing. As we show later the number of contributions peaked in early 2004, and is decreasing since then. However, it cannot be deduced from this data that the TIPs are responsible for that.

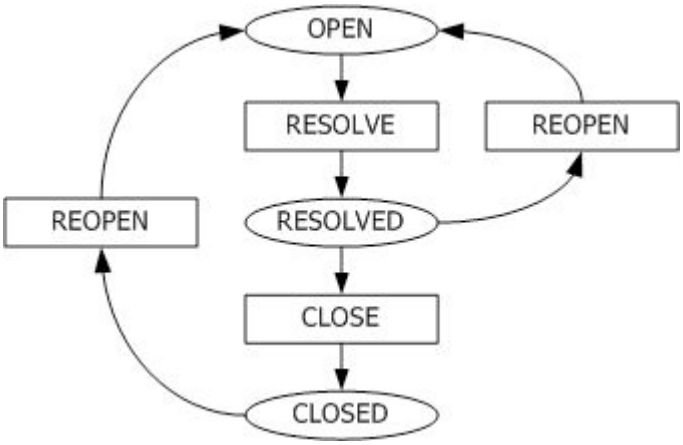

Figure 2: OpenACS Bug Tracker Workflow

age maintainer can resolve the bug, maybe by choosing one of the provided patches. The original submitter of a bug is the person who has to close it.

Note that the only quality assurance step is actually the last step, where the bug-submitter closes the issue. In practice it turns out that there is a high number of packages available, where for some of these packages the responsibility seems low, since many reported problems remain open. Also Villa [43] stresses the importance of closing old bugs. The bug-tracker contains currently 2896 entries, of which 1589 are closed, 422 are marked as resolved, and 886 are open. It seems as if the bug submitters care more about bug fixes than about closing the bugs they have opened.

3.4 Code Contributions

At the time of this writing, the CVS repository of OpenACS contains more than 2.5 million lines of code (mostly Tcl, SQL, HTML-templates and documentation, see Figure 3). In terms of logical units the OpenACS repository contains more than 200 packages.

In the following we present an analysis of the contents of the CVS repository to provide empirical evidence about the development. The CVS repository contains OpenACS 4.x and 5.x. For these versions 107 distinct committers have contributed to the CVS repository. 51 (48%) contributors can being classified as volunteers (non-profit contributors), and 54 (50%) have as well a commercial motivation (for profit contributors, regularly or at least on several occasions receiving payment for contributions of code to the OpenACS project or working for a company offering OpenACS support) and therefore we classify these as commercial. Two committers remained unclassified. The contributions of the classified contributors account for 99.97% of the total amount of contributions (see Figure 4). Since the unclassified contributors have little significance, we

Figure 3: Timeline of the Development of the Code Base.

left these out for the further analysis.

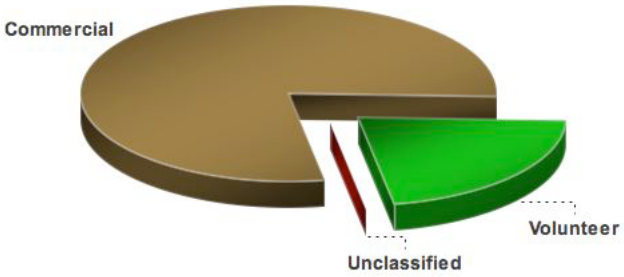

Figure 4: Total Number of Contributions.

Certainly, the distinction is not easy to draw, not at least since the status of a contributor might shift.[4] People receiving salaries for maintaining sites based on OpenACS, but not for participating in developing OpenACS, have been classified as volunteers. In comparison to the results of several surveys as described above [26, 11, 18, 21], the amount of commercial background within the OpenACS team is above the mean.

The top 15 OpenACS developers have contributed 72% of the total changes, 80% of these developers are rated as commercial. This highly skewed distribution is in line with findings from other studies of open source projects: A case study of the Apache project showed that there 88% of the code was developed by the top 15 programmers [28], where in the GNOME project, the top 15 pro-

[4]The classification was performed by the authors who are OpenACS contributers, based on project knowledge and Internet recherche.

grammers were responsible for "only" 48% of the code [25]. A similar distribution for the lines-of-code contributed to the project was found in a community of Linux kernel developers [21]. Also the results of the Orbiten Free Software survey [12] are similar, the first decile of programmers was responsible for 72%, the second for 9% of the total code. In a set of 8,621 SourceForge.net projects, the top decile was found to be responsible for 79% of the contributions[24].

We measure the contributions of the community members by the number of acts where the community members have committed content to the code base (checking in a file). About 104,000 contributions have been performed over the lifetime of the repository. A code contribution can be either a *checkin* (providing initial code) or a *modification* or a *removal* of a file. Using the policy-governed text in changelog messages[5], an additional classification of different contributions has been performed: This resulted in about 1,100 contributions pertaining to a TIP (1.1% of the total), 8,400 being bug fixes (8% of the total) and 1,700 patches (1.6% of the total), defined as being code committed for someone else without commit privilege. As this number of patches is relatively small, and the background of the original programmer is unknown, these would not have a significant effect on the relation between volunteers and commercial contributors, and this effect is therefore neglected.

	Contributions	Number	Ratio
Commercial	81,828	54	1,515
Volunteer	22,691	51	445

When we compare the non-profit committers with the commercial ones, we see that the number of contributions of a commercial commiter is more than three time higher than the contributions of a volunteer. This reflects well the structure of the OpenACS where the initial development was performed by the company ArsDigita. Also after the end of ArsDigita packages are frequently developed by companies for profit. Also, professional full time developers can spend often more time on developing a system than volunteers. Using a rank-based Mann Whitney U-test ascertains (at $p < 0.05$) that the commercial group leads in lines-of-codes, more TIP-related contributions, more bug fixes, more patches and more different packages worked on.

By distinguishing the contributions between code, documentation and others, while the commercial group is responsible for 78% of contributions of source code, this difference is even more pronounced in their efforts in code documentation with 82%. They also dominate in the group of TIP-related contributions, where commercial committers are responsible for 89% compared to 78% for non-TIP-related, and for patches, where they are responsible for 87%. This effect is not visible for bug fixes, where the percentage is mostly even (78% for non-bug fixes compared to 76%).

[5]Quoting: *CVS commit messages and code comments should refer to bug, TIP, or patch number if appropriate, in the format "resolves bug 11", "resolves bugs 11, resolves bug 22", "implements TIP 42", "implements TIP 42, implements TIP 50", "applies patch 456 by User Name", "applies patch 456 by User Name, applies patch 523 by ...".*

3.5 Analysis by Types of Packages

Next, the contributions for different types of packages are analyzed. OpenACS provides a division between kernel packages providing the general infrastructure and application packages, where .LRN is an important subgroup. For all three categories we distinguish further between optional and non-optional packages. The results are summarized in Figure 5 showing the contributions of the two contributor groups by package type.

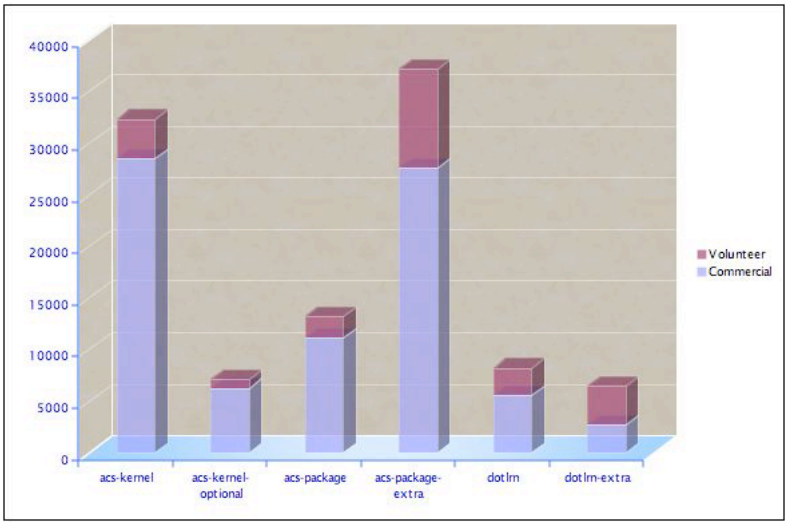

Figure 5: Contributions per Package Type.

It is interesting to see that the non-profit developers account for only 11% of the changes in the kernel, whereas in the code of the application packages or for the .LRN-components the contributions is much stronger (e.g. 57% in .LRN-extra). This can be explained by the strong usage of .LRN on universities worldwide, contributing code developed to satisfy their needs.

Furthermore, we observed that kernel packages score significantly higher in almost all dimensions (Mann Whitney U-test, $p < 0.01$) than application packages: They have more committers (both commercial and volunteer), more contributions ($p < 0.05$), more bug-fixes and also TIP-related contributions and patches. Interestingly, the difference in contributions, TIP-related contributions and bug fixes by volunteers is not significant. Also the percentage of commercial committers within the packages is not different between kernel and application packages, as is the amount of activity, measured in contributions per day of lifetime, i.e. since the initial checkin, although the lifetime itself, and the number of months there was work on the packages, do differ significantly and are larger for kernel packages.

Another point to explore is whether a dominance of commercial committers has any effect on other attributes of packages. We find that the higher the

percentage of commercial background is, measured either by the percentage of committers or contributions, the lower the activity is (Spearman correlation, $p < 0.01$). This might be an indication of a form of development in which new packages are created by commercial committers, checked in and seldomly changed later on. This is also underlined by a significant negative correlation (-0.375, $p < 0.01$) between the percentage of commercial background and the number of months in which contributions to a package were performed. As there is also a significant correlation to overall lifetime, we computed a stepwise linear regression with numbers of active months as dependent variable. Lifetime of a package alone reaches an R^2-adjusted of 0.338, including the percentage of commercial background leads to significant increase to an R^2-adjusted of 0.471 and has a negative coefficient, supporting the hypothesis. Also the standard deviation of programmers active within a month with any activity decreases with the amount of commercial background within a package (-0.577, $p < 0.01$). Concluding, we see that a large proportion of commercial background in a package leads to a low number of total and volunteer developers in this package (-0.473 resp. -0.739, $p < 0.01$), low activity and small variations in number of active developers between the periods of activity. It seems that commercial developers tend to contribute these packages, maintain them mostly on their own and only seldomly, maybe depending on receiving respective mandates. Therefore this form of sideways development seemingly often does not progress in the postulated 'open source' way, but might constitute a different development mode.

3.6 Changes of Contributions over Time

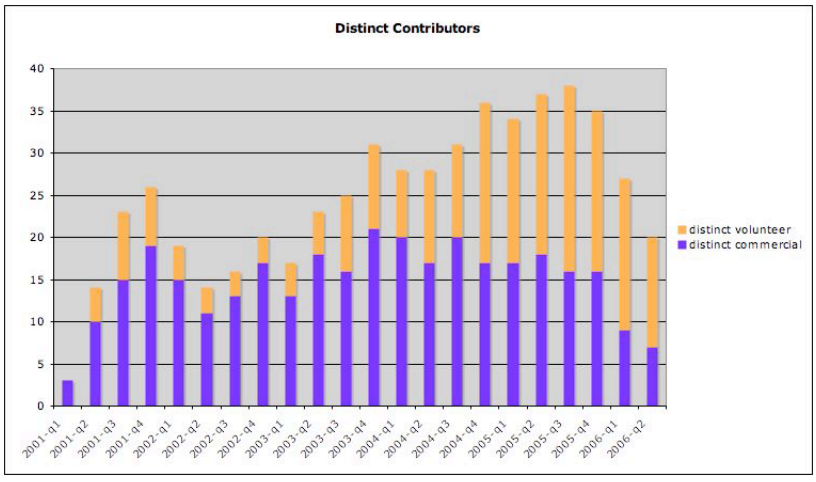

Figure 6: Distinct Contributors.

The analysis of the CVS data over time shows the shift from primarily com-

mercial contributors to more and more volunteer contributers. As shown in Figure 6 the number of non-profit contributors is constantly growing, while the number of commercial contributors reached its peak at the end of 2003. It is also interesting to see that although more than 100 contributors account for the project, there was no quarter year where more than 38 people have contributed to OpenACS so far. When we look at the number of contributions instead of the number of contributers, we see that number of contributions of the volunteers growing and big changes in the contributions of the commercial group (see Figure 7). The peaks in this diagram reflect the contributions to the major releases of OpenACS and .LRN.

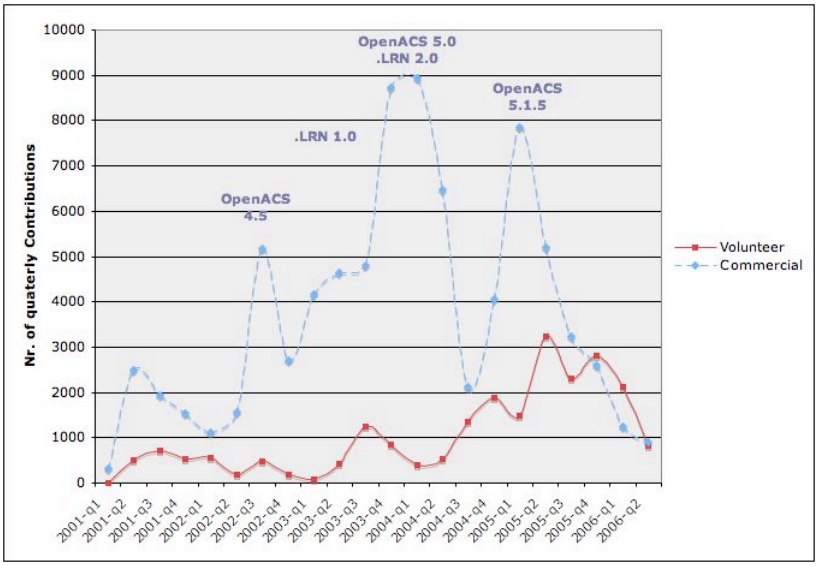

Figure 7: Number of Contributions over Time by Type of Contributor.

Both of these diagrams hint at the fact that this dominant position of the commercial group might erode over time. However, in terms of productivity, it takes several volunteer developers to replace one commercial developer. Currently the development of OpenACS is performed with quarterly about 1,700 contributions where the peak rate was in the 4th quarter of 2003 with nearly 10,000 contributions. For comparison, in the early phases of the GNOME project (1997-1999), a mean number of contributions per quarter of around 30,000 with peak rate of 38,000 was found [25], the mean within a set of 8,621 SourceForge.net projects was around 600 [24]. Regarding the number of participants, 354 distinct contributors were found in an analysis of the CVS repository of FreeBSD [7], nearly 400 for Apache and 486 for the Mozilla project [28].

4 Conclusion

In this paper, we have detailed OpenACS and its community as a case study of a project between commercial interests, securing investments, and technical development. The complex history has shaped both the community itself, and the practices adopted. We have found that indeed developers with a commercial interest dominate the history and code base of OpenACS, but that this fact might be slowly changing. This large amount of commercial interest in the project has led to a governance structure which puts great value on control and stability by requiring Technical Improvement Proposals for major changes. On the other hand, this rigidity seems to have affected the way of work, in that sideway developments might be established creating coexisting sub-frameworks. From an architectural viewpoint, this would be disadvantageous, and it might also have the effect of preventing true 'open source' style development, as the code in these parts would tend to be more specific and only usable in a certain context. In the empirical data, there seem to be indications for this happening especially in conjunction with commercial developers: We have found that packages being to a high degree dominated by commercial background tend to include less developers overall and less volunteers, and also tend to be changed less often and by the same group of people. If this trend would be continuing and increasing, a series of mostly isolated 'islands' could result.

In this respect, OpenACS, with its early and heavy involvement from commercial interests might prove a testbed for developments possibly taking place in several open source projects. It will be an important issue in the future, how the different interests of volunteer and commercial contributors in such projects can be aligned, and how the community is able to cope with demanding changes such as market forces of Web2. Through the strong investments of companies like ArsDigita and the highly flexible framework approach OpenACS has started with an advantage over competing projects. Over the last years, both the interest in collaborative web environments but as well the competition in this area increased. Without any doubt, the community, the structures and processes, and the product itself will and must continue to evolve and to adapt to new and changing requirements and situations. However, the project has come too long a way to die out, and its existence and continuation is ensured by a remarkable conglomeration of interests among companies, NPOs, and volunteers.

References

[1] G. Alberer, P. Alberer, T. Enzi, G. Ernst, K. Mayrhofer, G. Neumann, R. Rieder, and B. Simon. The learn@wu learning environment. In *Proceedings of Wirtschaftsinformatik 2003, 6th International Conference on Business Informatics*, Dresden, Germany, September 2003.

[2] J. Aufrecht. TIP #61: Guidelines for cvs committers. OpenACS Improvement Proposals, http://openacs.org/forums/message-view?message_id=185506.

[3] B. Behlendorf. Open source as a business strategy. In C. DiBona, S. Ockman, and M. Stone, editors, *Open Sources: Voices from the Open Source Revolution*. O'Reilly and Associates, Cambridge, Massachusetts, 1999.

[4] L. Belady and M. Lehman. A model of large program development. *IBM Systems Journal*, 15(3):225–252, 1976.

[5] F. Bergmann. Who is using OpenACS. OpenACS Q&A, http://openacs.org/forums/message-view?message_id=352641.

[6] B. Berliner. Cvs ii: Parallelizing software development. In *Proceedings of the 1990 Winter USENIX Conference*, pages 341–352, Washington, D.C., 1990.

[7] T. T. Dinh-Trong and J. M. Bieman. The freeBSD project: A replication case study of open source development. *IEEE Transactions on Software Engineering*, 31(6):481–494, June 2005.

[8] J. Erenkrantz. Release management within open source projects. In *Proceedings of the 3rd Workshop on Open Source Software Engineering, 25th International Conference on Software Engineering*, pages 51–55, Portland, Oregon, 2003.

[9] K. Fogel. *Open Source Development with CVS*. CoriolisOpen Press, Scottsdale, Arizona, 1999.

[10] M. J. Gallivan. Striking a balance between trust and control in a virtual organization: A content analysis of Open Source software case studies. *Information Systems Journal*, 11(4):277–304, 2001.

[11] R. A. Ghosh. Understanding free software developers: Findings from the floss study. In J. Feller, B. Fitzgerald, S. A. Hissam, and K. R. Lakhani, editors, *Perspectives on Free and Open Source Software*, pages 23–46. MIT Press, Cambridge, MA, 2005.

[12] R. A. Ghosh and V. V. Prakash. The Orbiten free software survey. *First Monday*, 5(7), July 2000.

[13] K. Gilroy. Collaborative e-learning: The right approach. *ArsDigita Systems Journal*, March 2001.

[14] P. Greenspun. Introduction to AOLserver. *LinuxWorld*, September 1999.

[15] P. Greenspun. *Philip and Alex's guide to Web publishing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[16] M. Hahsler and S. Koch. Discussion of a large-scale open source data collection methodology. In *Proceedings of the Hawaii International Conference on System Sciences (HICSS-38)*, Big Island, Hawaii, 2005.

[17] J. Hamerly, T. Paquin, and S. Walton. Freeing the source: The story of Mozilla. In C. DiBona, S. Ockman, and M. Stone, editors, *Open Sources: Voices from the Open Source Revolution*. O'Reilly and Associates, Cambridge, Massachusetts, 1999.

[18] A. Hars and S. Ou. Working for free? - Motivations for participating in Open Source projects. In *Proceedings of the 34th Hawaii International Conference on System Sciences*, Hawaii, 2001.

[19] R. E. Hawkins. The economics of open source software for a competitive firm - why give it away for free? *NETNOMICS: Economic Research and Electronic Networking*, 6(2), 2004.

[20] F. Hecker. Setting up shop: The business of open-source software. *IEEE Software*, 16(1):45–51, January/February 1999.

[21] G. Hertel, S. Niedner, and S. Hermann. Motivation of software developers in open source projects: An internet-based survey of contributors to the Linux kernel. *Research Policy*, 32(7):1159–1177, 2003.

[22] J. Holck and N. Jorgensen. Do not check in on red: Control meets anarchy in two open source projects. In S. Koch, editor, *Free/Open Source Software Development*, pages 1–26. Idea Group Publishing, Hershey, PA, 2004.

[23] N. Jorgensen. Putting it all in the trunk: Incremental software engineering in the FreeBSD Open Source project. *Information Systems Journal*, 11(4):321–336, 2001.

[24] S. Koch. Profiling an open source project ecology and its programmers. *Electronic Markets*, 14(2):77–88, 2004.

[25] S. Koch and G. Schneider. Effort, cooperation and coordination in an open source software project: Gnome. *Information Systems Journal*, 12(1):27–42, 2002.

[26] K. R. Lakhani and R. G. Wolf. Why hackers do what they do: Understanding motivation and effort in free/open source software projects. In J. Feller, B. Fitzgerald, S. A. Hissam, and K. R. Lakhani, editors, *Perspectives on Free and Open Source Software*, pages 3–22. MIT Press, Cambridge, MA, 2005.

[27] C. Meeks and R. Mangel. The arsdigita community system education solution. *ArsDigita Systems Journal*, September 2000.

[28] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of Open Source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, 2002.

[29] n.a. Homepage of OpenACS. http://www.openacs.org/.

[30] n.a. Homepage of ProjectOpen. http://www.project-open.com/.

[31] n.a. Homepage of the AOLserver project. http://www.aolserver.com/.

[32] n.a. Homepage of the .LRN project. http://www.dotlrn.org.

[33] S. O'Mahony. Guarding the commons: How community managed software projects protect their work. *Research Policy*, 32(7):1179–1198, 2003.

[34] J. Ousterhout. Free software needs profit. *Communications of the ACM*, 42(4):44–45, April 1999.

[35] J. K. Ousterhout. Tcl: An embeddable command language. Technical Report UCB/CSD-89-541, EECS Department, University of California, Berkeley, 1989.

[36] J. K. Ousterhout. Scripting: Higher-level programming for the 21st century. *Computer*, 31(3):23–30, 1998.

[37] Per Cederqvist et al. *Version Management with CVS*. Network Theory Ltd, Bristol, UK, 2002.

[38] E. S. Raymond. *The Cathedral and the Bazaar*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1999.

[39] G. Recco. Who is using OpenACS. OpenACS Q&A, http://openacs.org/forums/message-view?message_id=352641.

[40] G. Robles, S. Koch, and J. M. Gonzalez-Barahona. Remote analysis and measurement of libre software systems by means of the CVSanalY tool. In *ICSE 2004 - Proceedings of the Second International Workshop on Remote Analysis and Measurement of Software Systems (RAMSS '04)*, pages 51–55, Edinburgh, Scotland, 2004.

[41] I. Samoladas, I. Stamelos, L. Angelis, and A. Oikonomou. Open source software development should strive for even greater code maintainability. *Communications of the ACM*, 47(10):83–87, October 2004.

[42] J. B. Tran, M. W. Godfrey, E. H. Lee, and R. C. Holt. Architectural repair of Open Source software. In *Proceedings of the 2000 International Workshop on Program Comprehension (IWPC'00)*, Limerick, Ireland, 2000.

[43] L. Villa. Large free software projects and bugzilla. In *Proceedings of the Linux Symposium*, pages 471–480, Ottawa, Canada, 2003.

[44] Y. Ye, K. Nakakoji, Y. Yamamoto, and K. Kishida. The co-evolution of systems and communities in free and open source software development. In S. Koch, editor, *Free/Open Source Software Development*, pages 59–82. Idea Group Publishing, Hershey, PA, 2004.