Leveraging Open-Source Software in the Design and Development Process

Collin Green
NASA Ames Research Center
M/S 262-4
Moffett Field, CA 94035 USA
collin.b.green@nasa.gov

Alex Eiser
NASA Ames Research Center
M/S 262-4
Moffett Field, CA 94035 USA
alex.eiser@nasa.gov

Irene Tollinger
NASA Ames Research Center
M/S 262-4
Moffett Field, CA 94035 USA
irene.tollinger@nasa.gov

Lanie Castro
NASA Ames Research Center
M/S 262-4
Moffett Field, CA 94035 USA
lanie.b.castro@nasa.gov

Christian Ratterman
NASA Ames Research Center
M/S 262-4
Moffett Field, CA 94035 USA
christian.d.ratterman@nasa.gov

Alonso Vera
NASA Ames Research Center
M/S 262-4
Moffett Field, CA 94035 USA
alonso.vera@nasa.gov

Guy Pyrzak
NASA Ames Research Center
M/S 262-4
Moffett Field, CA 94035 USA
guy.pyrzak@nasa.gov

CHI 2009, April 4–9, 2009, Boston, Massachusetts, USA
ACM 978-1-60558-247-4/09/04.

Abstract
This paper presents a case study of the NASA Ames Research Center HCI Group's design and development of a problem reporting system for NASA's next generation vehicle (to replace the shuttle) based on the adaptation of an open source software application. We focus on the criteria used for selecting a specific system (Bugzilla) and discuss the outcomes of our project including eventual extensibility and maintainability. Finally, we address whether our experience may generalize considering where Bugzilla lies in the larger quantitative picture of current open source software projects.

Keywords
Open-source software, software development, collaboration, benefits analysis.

ACM Classification Keywords
D.2.13 [Software Engineering]: reusable software, reuse models; H.5.2 [Information Interfaces and Representation]: prototyping, user-centered design; K.5.1 [Legal Aspects of Computing]: licensing.

Introduction
The National Aeronautics and Space Administration (NASA) Constellation Program will span more than 30 years and will include manned missions to Earth orbit, the International Space Station, the Moon, and Mars. Constellation will replace NASA's aging Space Shuttle program over the next decade. Managing risk, including

having an understanding of current and historical problems is critical to mission success from the very outset. Throughout each mission, problems may endanger the lives of the crew and support staff, compromise mission objectives, or result in costly and time-consuming repairs. Preventing even one small problem can save time, money, and lives.

Our Human-Computer Interaction (HCI) Group was charged with developing a new Problem Reporting, Analysis, and Corrective Action System for the Constellation Program (CxPRACA). CxPRACA is intended to capture and store program-wide data on engineering problems and non-conformances so that problems can be understood (and any related risks can be mitigated).

We begin with a brief overview of the composition of our HCI group and some additional details about the context and domain of our project. Subsequently, we present a case study of our design and development of the CxPRACA system for NASA based on adaptation of the Bugzilla [3] open source bug tracker, including criteria we used to determine whether Bugzilla was a good candidate for this adaptation. We discuss, from an HCI and technical perspective, some of the tradeoffs involved in taking this approach, and also consider whether our experience is generalizable considering where Bugzilla lies in the larger quantitative picture of the current open source software (OSS) community.

A Software Development Challenge at NASA

The NASA Ames Research Center HCI Group

How can an HCI Group, working for a large government agency, have enough development capacity to bring their methods to bear on major mission systems? One way is to pair-up with a development group. The HCI Group does the requirements analysis, understanding the functional requirements based on acquiring domain expertise and the development team executes on those requirements. This is perhaps the more typical model in industry. It has been our experience that, although sometimes successful, it can also be the case that user requirements get pushed to the background by the software development team, focusing first on technical capabilities of the software. This produces software that fails to meet user needs and the product as a whole fails. Our aim in this effort was to have the HCI Group take control of, and responsibility for, the software development process while at the same time not becoming a development team disguised as an HCI team.

The HCI Group at NASA Ames is composed largely of HCI-trained people, along with a few developers and testers. The group has seventeen people out of which approximately five contribute to software process management, development and testing. It was therefore clear that we either had to hire a large number of developers or start from an existing application that would allow evolution by a small group of developers. HCI and usability groups in industry often operate as "guns for hire". Members of the group get deployed to development teams to contribute user requirements, interface designs, and end-user assessment. On the other hand, some companies, especially those that are not in the business of software development (e.g., in the financial and medical industries) have HCI people tasked with developing systems, often for internal use. It is to these sorts of applications that this case study should be particularly relevant.

Project Overview

The CxPRACA system we were asked to develop was a large, NASA-wide database-driven application. It would eventually have thousands of users and millions of records and need to evolve over the 30+ year planned lifecycle of the Constellation Program. It needed to be platform-independent and available to all ten NASA centers and to NASA contractors as well. The application domain can be characterized, most broadly, as engineering risk and safety management. The system holds information about software and hardware non-conformances. The system must support email, search, reporting, and change history tracking in addition to basic relational database functions. The CxPRACA system must also be very robust: it is important to in-flight mission operations and therefore requires 24/7/365 availability and must be highly secure (much of the data come to the government from private contractors and so are competitively sensitive and/or proprietary).

Adapting OSS for a NASA Application

Establishing System Requirements

Initially, our group conducted field research on problem reporting, focusing on both processes and systems, using contextual inquiry techniques [2]. The research covered existing NASA systems (at multiple centers) and systems used by the United States Navy, the Department of Energy (at nuclear power plants), and private corporations. Overall, the HCI Group spent over a year conducting user research to understand the requirements for the CxPRACA system before beginning to consider a technical solution (i.e., selection of a system).

The research identified three key problems:

- DATA FRAGMENTATION There existed many isolated PRACA systems with inconsistent data schemas (50+ on shuttle alone) resulting in poor support for search/trending.
- DATA INTEGRITY Information in PRACA systems was often incomplete due to records being entered after the analysis was complete. This, in turn, was due to a lack of support in the software tools for the tactical aspects of the work.
- DATA INTEGRATION Information in related (non-PRACA) systems was inaccessible. Data related to problem reporting (e.g., part numbers, part assemblies, engineering diagrams, etc.) was fragmented, incomplete and stored in disconnected systems such that data could not be associated with problem reports as required.

Mishap investigations of the Challenger explosion and the loss of Columbia called legacy PRACA systems dysfunctional and highlighted the need for a *single*, program-wide data set. Thus, software requirements for a new PRACA system developed to address these and many other issues were an Agency-wide priority. The HCI Group's task was to propose a software application that would meet key requirements and be available for production use within a short timeframe.

The team looked at a number of tools with the goal of finding a system that: 1) met the basic functional requirements that emerged from the key problems observed (i.e., had an analogous domain/usage model); 2) had sufficient configurability and extensibility to support tailoring across NASA centers; 3) was robust enough for large-scale production use, 4) was modifiable by NASA; 5) could be hosted on NASA hardware.

Open-Source Software and HCI

OSS has periodically been mentioned in the HCI literature, (e.g., [5] and [8]). However, it has primarily been discussed in the context of attempts to provide design work or design guidelines and though some authors have encouraged the HCI community to contribute to OSS projects, such collaboration has proven difficult in practice [1, 12].

There are at least three reasons for this difficulty. First, and most obviously, the OSS developer culture places value in the functionality of code more than in interaction design. Second, applying design to a project comprehensively is best completed prior to the start of development [8, 9]. OSS projects tend to be developed piecemeal by independent contributors, so comprehensive design efforts do not fit the development process well. Third, the tools used by OSS communities do not support usability work particularly well. For example, typical bug tracking applications and code-centric version-tracking systems do not offer users the ability to capture complex design ideas and discussions [14]. These facts have been somewhat discouraging to the HCI practitioner interested in working with OSS.

It was not our primary goal to improve the usability of any individual OSS project, nor did we seek to change the culture or tools used in the OSS community. Instead, our goal was to design and develop software more efficiently to suit NASA's needs by taking advantage of OSS as it currently exists. One way to incorporate OSS code into software development is to start with an OSS components-based framework, such as Java Eclipse, and build up a new application from these components. Another way is to adapt an already-built application that contains the features desired in the new application. In both cases, code reuse is intended to improve efficiency but the latter approach maximizes the amount of reused and shared code. Starting from components offers more flexibility but higher development cost because more code needs to be created. Starting from an existing application offers less flexibility but keeps costs lower.

However, starting from an existing system can offer more flexibility than one might imagine. Since the 1970s, software engineers have increasingly focused on modularity as a means to reduce the cost of changes and improve product quality [11]. Improvements in language structure and architectural conventions have lead to practices in modern software development that are focused on modularity of design. Considering these points (and other factors, as detailed below), we opted to adapt an already-built OSS project as the basis of our CxPRACA system.

Key Aspects of OSS System Selection

BASIC FUNCTIONAL REQUIREMENTS: ANALOGOUS DOMAIN

The HCI team had the view that problem tracking was not a NASA-unique domain. Any company that developed hardware or software would require a closed-loop system for capturing and resolving problems (from auto-makers to software companies). This drove us to look closely at analogous domains such as bug-tracking systems.

Though differences exist, software bug tracking and engineering problem reporting are clearly analogous. For both activities, a problem is reported and stored in a digital system so that the information can be shared with a community. In both cases, a plan for solving the

problem is collaboratively built and documented in the system. In addition, both types of systems allow work to be assigned, tracked, and its completion recorded digitally. In the long term, the records in both systems act as a historic account that can support the solution of new bugs/problems or analyzing and refining development and testing processes. This similarity was an important aspect of our decision.

We identified Bugzilla—an OSS bug tracker managed by the Mozilla Foundation—as a system that included the basic functionality needed for the CxPRACA system. The match between our required functionality for CxPRACA and that offered by Bugzilla was one important factor in our initial selection of Bugzilla.

We discuss below the other factors we considered: Bugzilla's configurability and extensibility, which supported rapid development; Bugzilla's robustness; and finally, Bugzilla's OSS license allowing modification of the code as much as needed.

CONFIGURABILITY AND EXTENSIBILITY

We define configurability as an application's built-in support for a user or administrator to make changes to the application (defaults, layouts, data captured, etc.) without any changes to code. Bugzilla had recently been augmented, by OSS developers, to support administrator-managed custom fields that can be added without manipulating the code and or doing manual database updates. Our team was able to configure the Bugzilla data-entry interface to support five times more data fields than are available in the standard Bugzilla install. This feature of Bugzilla was critical to our rapid and successful deployment of the initial CxPRACA system.

We define extensibility as support for development of new application features. We focus on two aspects of extensibility: abstraction of UI code from backend code and architecture. The abstraction of the UI and backend allowed us to make changes in one area without creating ripples and introducing bugs in the other. On the UI side, Bugzilla includes substantial support for skins and templates. In fact, the initial round of development focused on changes to the template files that control layout and visualization. The underlying data management layers and program logic layers remained largely untouched. Later, we added backend features including: new data field types, full text search, server-side data validation (e.g., user names, dates), and other improvements.

In terms of architectural extensibility, based on a documented API and a plug-in architecture, Bugzilla was actually sub-optimal. Bugzilla does have a documented API but does not fully support a plug-in architecture. This was a substantial negative feature of Bugzilla from our perspective as it meant that it was likely to cost more time to merge future versions of Bugzilla with the CxPRACA code. As discussed later in the Project Outcome Section, close collaboration with the OSS community, including contribution of code back to Bugzilla, has allowed us to deal with this issue.

Overall, the ease with which Bugzilla can be modified (through re-skinning, through parameter definition, and through administrator control of data schema and work processes) was important and allowed us to make relatively few code changes initially such that a production system could be deployed quickly.

Robustness

Robustness is the result of many different practices by a development group, whether in industry or the OSS community, from code reviews to comprehensive quality-assurance testing. We found that the QA processes the community follows lead to an overall quality level sufficient to make us confident that we would be able to spend our time improving its usability and capabilities rather than fixing bugs in the underlying code. This early impression has been borne out over the course of the almost two-year effort.

Modifiability

We define modifiability as the ability to access and change the application's code. Modifiability by NASA was one factor that pushed us toward the analysis of OSS applications. Depending on the tool, vendor provided software or Commercial Off-The-Shelf (COTS) tools usually have one of the following models of code control: 1) all code is only modifiable by the vendor, 2) part of the code is only modifiable by the vendor, 3) the code is source-available and can be modified. The applications in category three were mostly open source applications. We will talk in the Project Outcome Section below about how we were able to limit merging of features back into future releases based on the transparency of the OSS development process.

Related to modifiability, maintaining data on NASA servers is important from a data integrity and security perspective. There were tools that met many of the other criteria but not this one (e.g., the Google Base application which was closely tied to Google infrastructure and could not be hosted by NASA).

Through our analysis of the five above factors, we discovered two additional areas that turned out to be important. One was the accessibility of an active OSS development community and the other was the particular tool used to do distributed versioning by the community.

Accessibility of Active OSS Developers

Bugzilla's technical community is large for an OSS project. There were twelve active OSS developers working on Bugzilla, and the project's website listed over thirty companies (independent of the Mozilla Project) providing support, administration, hosting, and configuration/development for Bugzilla.

While we anticipated most of our development to be UI modifications, a proportion of our initial version did include new or modified features that required more extensive changes to Bugzilla (that proportion has increased in subsequent versions). Contact with an OSS developer proved very valuable. We retained the help of a senior Bugzilla developer to help us modify the Bugzilla backend for early versions of CxPRACA. Contact with an OSS developer also helped us return some of our development work to the OSS community for inclusion in standard releases (and for maintenance).

Distributed Version Control Scheme

One important factor in the smooth adaptation of Bugzilla was not inherent in the code itself, but rather was a feature of the Bugzilla developer community. Their preferred version tracking tools were compatible with development of independent (branched) versions of Bugzilla.

OSS projects promote a very open, flexible organizational development structure and as a result tend to employ robust version control systems. While a traditional version control system (e.g., CVS) allows a team to concurrently code the same application and commit to a code base without collisions or broken code, such systems lack the ability to manage code from diverse and distributed independent codebases (as with OSS). Bugzilla uses an open-source version control system called Bazaar that can automate the merging of code from separate, previously branched code bases.

This model allowed our team to continue development of a separate version in multiple stages and to maintain control of our release schedule. The use of Bazaar (rather than a traditional version control system) allowed us to gain from (and contribute back to) the work of the larger Bugzilla community while maintaining our own (faster) development cycle.

Project Outcome

Bugzilla was selected as the base application for CxPRACA in October 2006 and Version 1.0 was released in January 2007. Over the next few months the team worked with the user community to demonstrate capabilities, gather feedback, work on refining the system, and explore new features. In May 2007, the first center to use the system was identified, Langley Research Center. In August 2007, the first users began entering records into the system. Since then the following centers have adopted the system as well: Dryden Flight Research Center, Kennedy Space Center, Glenn Flight Research Center. More locations and groups will start to use the system as more Constellation hardware goes into manufacture and test. CxPRACA will eventually have thousands of users and millions of records.

All the development work, including four major releases (January 2007, July 2007, December 2007, June 2008) has been accomplished in less than two years with two and later three developers. Each release is observed in context and usability tested with novice and experienced participants. While we identify additional usability issues for subsequent iterations, user feedback has generally been very positive and helpdesk call volume has been low (less than one per week).

In the end, the implementation of the CxPRACA system has turned out to be substantially different from the Bugzilla base from which we started. Beyond the UI and usability improvements, we implemented some substantial changes including: 1) improvements to search, 2) new field types (e.g., long text fields), 3) generalization of Bugzilla "core" fields (e.g., OS, platform, etc.), 4) extended capabilities to link data (between systems, records, and within a record), 5) new data structures (e.g., to support capture of multiple groups of fields on a single record).

Some features, such as search, field types, and generalization of core fields were incorporated back into the Bugzilla code base and are now maintained by the Bugzilla OSS community. Also, the HCI Group contributed back the user-research in the form of code for those features. Once a new version of Bugzilla is released, new (NASA-developed) features, now embedded in Bugzilla, will replace the CxPRACA implementation of them. Lastly, there are certain features, such as the new data structure for capturing multiple groups of fields on a record, which do not map

well onto the bug-tracking domain and will be maintained directly by NASA. We have worked to stay aligned with Bugzilla and have managed to limit divergence such that approximately 90% of our application's code is core Bugzilla and only 10% is NASA-maintained (see Figure 1).

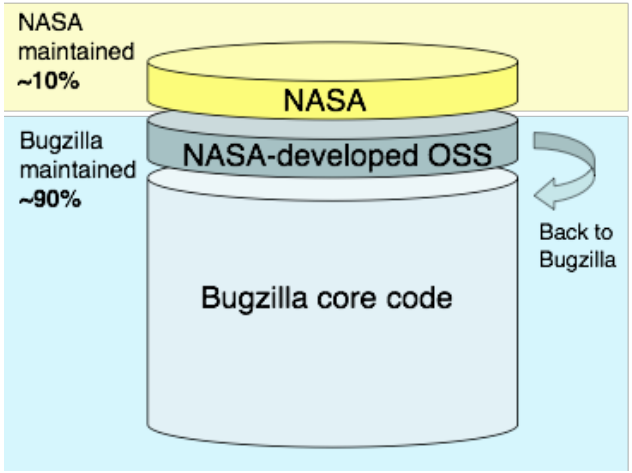

Figure 1. Although CxPRACA as a whole is not open source, some of the code developed by NASA for CxPRACA has been open-sourced and released as part of core Bugzilla under the Mozilla public license.

A significant advantage of having used OSS is that our team can leverage new versions and updates without much development work. After our adoption of Bugzilla, new versions have provided significant features (e.g., a fully customizable workflow) that CxPRACA required and the HCI Group did not have to implement. We are seeking to maximize the likelihood of seamless integration of CxPRACA with new versions of Bugzilla. Our team has actively engaged with the larger Bugzilla community. We contribute developer time to bug reporting, bug fixes, and feature implementation in the core Bugzilla codebase. This improves our developers' knowledge of the system and increases their standing in the community. Although this strategy will cost development resources, it will reduce the cost of maintenance and new feature development for our project.

An unanticipated outcome of the successful development of a NASA-wide PRACA system was the request from Constellation management to explore the possibility of developing other systems using the same code base. Over the past year, this has in the design and development of three other systems involved in risk management of space missions. 1) A Hazard Analysis database that catalogues antecedent understanding of potential hazards in both hardware and software. 2) A Failure Mode Effects Analysis System that captures predictive analyses of how software and hardware can fail, especially as complex systems are integrated. 3) A database that captures information from each mandatory inspection that the government carries out of contractor development activity. For each of these systems, the NASA Ames HCI Group conducted an analysis of user requirements and determined that, with minor extensions, the functionality of the Bugzilla-based code could support the requirements. Critically, this could be done without branching the code base (now underlying four systems), as this would have significantly increased the development cost.

In the past few months, the HCI Group has received numerous requests to use the system outside of the Constellation program. The International Space Station

program has approved a migration of two systems to the Bugzilla-based solution by February 2009. Additionally, the Space Shuttle Orbiter Project will be synchronizing data from existing systems each night to take advantage of superior search and trending capabilities by November 2008. These are significant achievements in that the system will go from supporting the nascent Constellation program to supporting operational vehicles carrying human crews 24 hours a day, 7 days a week (in the case of ISS).

Is Bugzilla a Special Case?

With few developer resources, we successfully adapted Bugzilla as the basis for our CxPRACA system. A number of features of Bugzilla (detailed above) contributed to this success. Would other OSS projects be amenable to such adaptation? Can our experience serve as a model for other HCI practitioners or is Bugzilla unique among OSS projects? In this section, we consider whether Bugzilla may be a special case within the space of OSS projects by looking at that space as a whole and placing Bugzilla into context.

The Current State of OSS and the OSS Community

As a prominent aggregator of OSS projects, SourceForge [13] has been analyzed by studies focusing on a variety of HCI topics [4, 7]. However, specific limitations of that data set have been noted [6]. For example, SourceForge is the "repository of record" for larger projects but often not the "repository of use", which means that some information about projects is incorrect (e.g., an active project shows zero activity, zero commits, zero open bugs, etc.). As an alternative, we chose to use Ohloh.net [10]—another OSS aggregator—because it links directly to the tracked projects' "repositories of use" and thus provides a more accurate and up-to-date data set. In addition, Ohloh.net made available historical data about each project that allowed us to examine growth in OSS projects.

Ohloh.net tracks a large number of OSS projects and keeps statistics on the size and maturity of their codebases as well as the size and activity level of their developer communities. The numbers of OSS projects and developers have grown dramatically in recent years. The data indicate that a large number of active OSS projects exist: statistics were gathered on 7,824 separate projects (data were current as of September 12, 2007). This number has grown particularly rapidly in the last 10 years. Figure 2 shows the number of OSS projects over the last 37 years, as indicated by their starting dates on Ohloh.net. The growth seems to follow a power law. The most notable feature of this figure is the massive growth in OSS activity since around 1996. This means that an HCI practitioner is likely to have multiple projects from which to select when seeking to adapt OSS to a new target domain.

Many OSS projects have begun only recently (the newest recorded OSS project was a mere 11 days old). The average age of the projects listed on Ohloh.net was 1,128 days (just over three years old). The oldest project in the data set was 13,768 days old (a project called Magnolia has a recorded beginning in 1970). The mean age of projects is somewhat misleading because the distribution is highly skewed, with a long tail representing a small number of older projects. The median age of OSS projects was 925 days (about two and a half years old) and the mode age was 588 days (just over a year and a half old). The distribution of project ages is shown in Figure 3. As noted above, a

large number of projects have begun in the last decade or so (a large number of projects are under 3,000 days old), and this pattern reflects substantial recent growth in OSS as a development model and technological trend.

Of course, it is also reasonable to consider the sizes of the codebases being produced during the lifetime of a project. Here, we use lines of code (LOC) as a measure of project size. (Note that this measure can be artificially inflated if developers check libraries, utilities, or other existing code into their development repositories). OSS projects listed on Ohloh.net ranged in size from 2 LOC to over 25.3 million LOC (the Debian/GNU Linux operating system has a codebase generated by at least 520 developers over the course of nearly a decade). The mean project size was 125,701 LOC, but again it is prudent to look at another measure of central tendency to better understand the data; the median project size was 14,980 LOC. The distribution of project sizes is shown in Figure 4 (note the logarithmic scale).

The bulk of OSS projects have codebases that measure in the tens of thousands of lines of code. Naturally, some of the variance in the size of OSS codebases is attributable to the variety in projects (some are plug-ins to other applications while others constitute complete operating systems).

Because OSS projects are not constrained by traditional notions of hiring and firing paid developers, it is also interesting to examine the size of the development teams working on such projects.[1] Ohloh.net listed projects that ranged from zero active developers (where active developers are those that have committed code in the last 12 months) to 1,990 active developers (the large team working on Linux Kernel 2.6). The mean number of developers on a project was 4.51, while the median and mode were both 1. The distribution of active development team sizes (and total developers, not just those active in the last twelve months) is illustrated in Figure 5.

One notable feature of this distribution is that there are many small development teams (fewer than 15 active developers) and relatively few larger development teams. This pattern is reflected in the distribution of total developers (the gray line in Figure 5).

In general, the data provided by Ohloh.net reflect an OSS community that is large, active, and growing. OSS projects come in all sizes, and typically have small development teams. (However, one would be correct in assuming that larger projects tend to have more developers: the correlation between project size [LOC] and active developers is positive, $r^2 = 0.40$).

Bugzilla Compared to Other OSS Projects

On the question of whether Bugzilla is somehow special among OSS applications, we can summarize by saying that Bugzilla is more mature than the average OSS project. Bugzilla's age, codebase, and development team size are noted in Figures 3, 4, and 5 to show where this project lies in the context of the larger space of OSS projects. Bugzilla (~8 years old) is older than

[1] There are, admittedly, other issues that arise with OSS. OSS projects are often maintained by volunteers, which can affect development in interesting ways: for instance, the direction and intensity of development may not be centrally controlled. Further, many large OSS projects (e.g., Mozilla Firefox) include teams of paid developers.

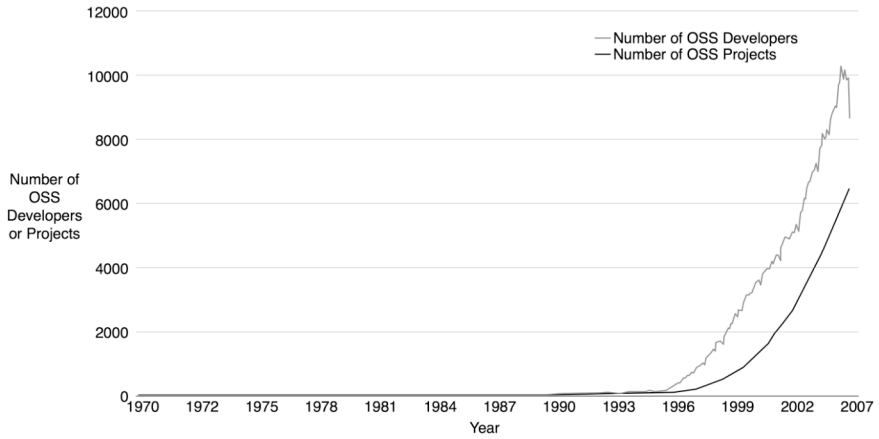

Figure 2. Growth of the OSS community (1970-2007).

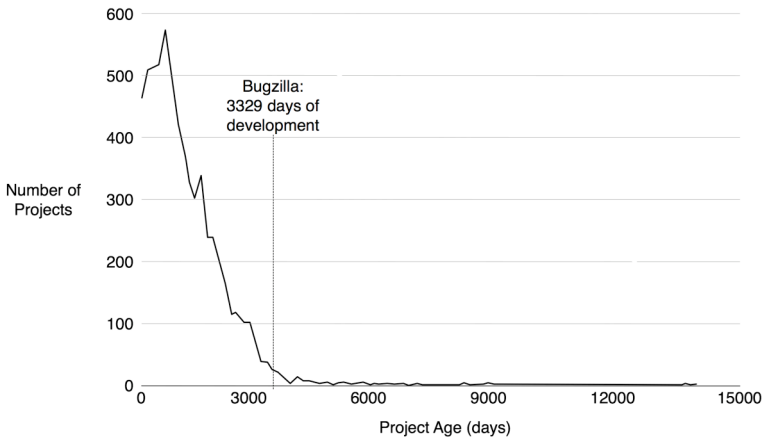

Figure 3. Frequency distribution of the ages of OSS projects tracked by Ohloh.net.

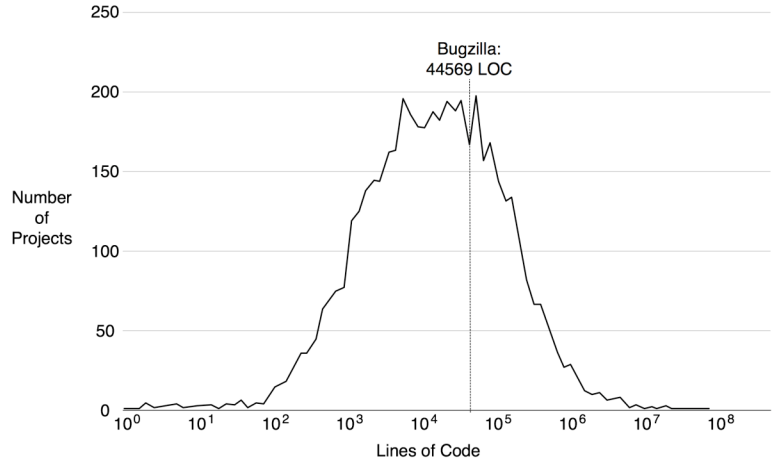

Figure 4. Frequency distribution of the number of lines of code in OSS projects tracked by Ohloh.net.

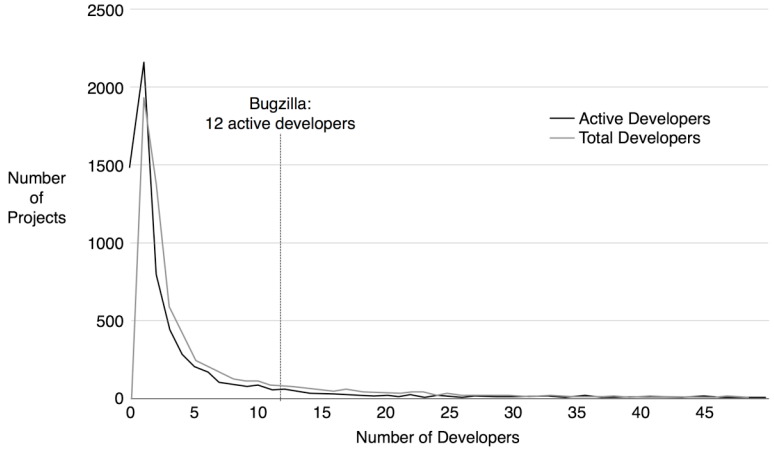

Figure 5. Frequency distribution of the number of developers on OSS projects tracked by Ohloh.net.

98% of the OSS projects surveyed, it has a larger codebase (44,569 LOC) than 68% of the projects surveyed, and its active development team (12 developers) is larger than 92%. Bugzilla is a substantial OSS project with a large, stable codebase that has been developed and tested over a long period by numerous developers. These facts (and Bugzilla's score on other criteria) made it an attractive candidate for our project.

That said, Bugzilla is not a conspicuous outlier in the space of OSS projects. There exist many comparably mature OSS projects: the Ohloh.net data listed at least 33 other OSS projects that were older than Bugzilla with larger codebases and a larger number of active developers; more than 450 projects had larger development teams. HCI groups that are considering in-house development of software should be encouraged to look to the OSS community for opportunities to adapt existing OSS projects, but they should take care to develop appropriate criteria for selecting a particular OSS project for adaptation.

Finally, it is worth noting that at least one peripheral factor worked in our favor when it came to accessing development support in the OSS community: the local area (Mountain View, CA) is very active in OSS development and has a rich technology community.

Conclusion

It is our position that HCI practitioners can realistically consider adapting OSS projects as the basis for their own specialized applications. Doing so allows an HCI team to play a substantial role in the development process with only a small investment in development resources. In addition, the number and diversity of currently active OSS project means that HCI practitioners have a far better chance of finding one that is compatible with their functional requirements than they had in the past.

The data from Ohloh.net illustrate that OSS has come into its own as a model for software development. The number of mature and well-known deployed OSS applications should also reassure HCI practitioners that this approach is worth considering for projects both large and small. However, taking the time to evaluate OSS projects in terms of core functionality, configurability and extensibility, robustness, modifiability, and activity of developer community may well lead an HCI practitioner to excellent candidates for adaptation.

Our own experience highlights this fact: in under two years, our HCI group was able to gather requirements and deploy multiple production systems by adapting the Bugzilla bug-tracking system to NASA's needs. Looking ahead, the adaptation of OSS has the potential to reduce the cost and effort required to maintain and improve these new NASA systems: the core code (that supports data management, search, etc.) will continue to improve through contributions from the OSS community leaving our team free to work on those features of the system that are specific to our target domain.

The gains from such collaboration are not one-sided, however. The OSS community stands to gain from HCI adaptation of OSS projects as well. First, an increase in the number of contributors is likely to further strengthen the OSS movement. Second, HCI involvement in particular will serve to improve the

usability of OSS projects as a by-product of usability enhancements for the new domain-specific application: new code, which implements usability and functional improvements, may be contributed back to the original OSS project and community.

Indeed, while modifying the Bugzilla system we made a number of usability improvements to the core system functionality (e.g., a redesign of the report creation interaction) that were determined by OSS developers to be valuable to the entire Bugzilla user base. By contributing these features back, we are able to share the long-term management of the feature as well as contribute better usability to a community with less HCI focus. This is the standard, valued approach for providing improvements within the OSS community. It is also valuable for the HCI product development team because the community takes collective responsibility for maintenance of new features that are accepted into the project's core codebase and that reduces the work required to integrate with future OSS releases.

In general, collaboration between HCI and OSS has the potential to be mutually beneficial. Our goal in this paper has been to describe how we collaborated with an OSS project without demanding that the OSS community change its current practices. Armed with a realistic picture of the OSS world, and with some criteria upon which to evaluate OSS projects for adaptation, HCI practitioners will (we hope) be able to build on our positive experience working with OSS.

Acknowledgments.
This work was supported by the NASA Exploration Systems Mission Directorate. We thank Robin Luckey and everyone at Ohloh.net for helping us obtain OSS data and for tolerating our traffic on the Ohloh.net server. We also thank Max Kanat-Alexander for his significant contribution to NASA's CxPRACA system.

References

[1] Benson, C., Muller-Prove, M., Mzourek, J. (2004). Professional usability in open source projects: GNOME, OpenOffice.org, NetBeans. Proc. CHI 2004.

[2] Beyer, H. & Holtzblatt, K. (1997). Contextual Design: A Customer-Centered Approach to Systems Designs. Morgan Kaufmann.

[3] http://www.bugzilla.org/

[4] Chengalur-Smith, S., and Sidorova, A. (2003). Survival of open-source projects: A population ecology perspective. Proc. of 24th International Conference on Information Systems (ICIS '03).

[5] Frishberg, N., Dirks, A. M., Benson, C., Nickell, S., Smith, S. (2002). Getting to Know You: Open Source Development Meets Usability. Proc. CHI 2002.

[6] Howison, J., and Crowston K. (2004). The perils and pitfalls of mining SourceForge. Proc. of the 1st International Workshop on Mining Software Repositories (MSR 2004), 7-11.

[7] Krishnamurthy, S. (2002). Cave or community? An empirical examination of 100 mature open source projects. First Monday, 7(6).

[8] Nichols, D. M., Thomson, K., and Yeates S. A., (2001). Usability and Open Source Software Development. Proc. of the Symposium on Computer Human Interaction (SIGCHI New Zealand), 49-54.

[9] Nichols, D. M., and Twidale, M. B. (2002). Usability and Open Source Software. Available online: http://www.cs.waikato.ac.nz/~daven/docs/oss-wp.html

[10] http://www.ohloh.net/

[11] Parnas, D. L. (1972). On the criteria to be used in decomposing system into modules. Communications of the ACM, 15(12), 1053–1058.

[12] Smith, S., Engen, D., Mankoski, A., Frishberg, N., Pedersen, N., and Benson, C. (2001). GNOME Usability Study Report. Available online: http://developer.gnome.org/projects/gup/usertesting.html.

[13] http://sourceforge.net/

[14] Twidale, M.B. and Nichols, D.M. (2005). Exploring Usability Discussions in Open Source Development. Proc. of the 38th Annual Hawaii International Conference on System Sciences (HICSS 2005).

[15] Wheeler, D. A. (2007). Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers. Available online: http://www.dwheeler.com/oss_fs_why.html.