ingimp: Introducing Instrumentation to an End-User Open Source Application

Michael Terry, Matthew Kay, Brad Van Vugt, Brandon Slack, Terry Park
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada
mterry@cs.uwaterloo.ca, {matthew.kay, bvanvugt, brandon.slack, t2park.uw}@gmail.com

ABSTRACT

Open source projects are gradually incorporating usability methods into their development practices, but there are still many unmet needs. One particular need for nearly any open source project is data that describes its user base, including information indicating how the software is actually used in practice. This paper presents the concept of *open instrumentation*, or the augmentation of an open source application to openly collect and *publicly disseminate* rich application usage data. We demonstrate the concept of open instrumentation in *ingimp*, a version of the open source GNU Image Manipulation Program that has been modified to collect end-user usage data. ingimp automatically collects five types of data: The commands used, high-level user interface events, overall features of the user's documents, summaries of the user's general computing environment, and users' own descriptions of their planned tasks. In the spirit of open source software, all collected data are made available for anyone to download and analyze. This paper's primary contributions lie in presenting the overall design of ingimp, with a particular focus on how the design addresses two prominent issues in open instrumentation: privacy and motivating use.

Author Keywords
Open source usability, avatars, personas, GIMP, OSS, free software, GPL

ACM Classification Keywords
H.5.2. Information interfaces and presentation: User Interfaces, evaluation/methodology.

INTRODUCTION

The open source software community has a rich set of tools at its disposal to support distributed software development. Source code repositories, bug tracking systems, and even tools as simple as "patch" all help to coordinate the activities of individuals who may never meet face-to-face. Importantly, these tools not only coordinate project members, they also open up the development process to the larger community: Individuals external to an open source project can download the source code, fix bugs they discover, submit patches, or simply report a bug, all without needing any special credentials or access rights to do so. The ability for *anyone* to participate is often cited as a potent catalyst for the creation of open source software [19].

While a mature set of tools scaffold open source software *development*, comparatively fewer tools exist to scaffold *usability* efforts in open source projects [1, 14]. To date, open source usability efforts have been primarily supported through general-purpose communication tools (e.g., mailing lists and blogs), repurposed development tools (e.g., Bugzilla), and a handful of resources, such as human interface guidelines (HIGs) [5, 9]. Collectively, these tools, along with an increasing number of project members dedicated to usability concerns, have done much to help bootstrap usability efforts in open source projects. However, a number of needs persist.

One of the unmet needs of the open source community is data describing the larger community of users: Who uses the software, for what purposes, with what level of expertise, in what types of computing environments, and so on. This descriptive data complements usability data by quantifying the community's actual day-to-day usage of the software, painting a picture of the user community not attainable by usability studies alone. For example, command usage counts can suggest which aspects of the interface are most important to the community, and which are not. Commercial software companies have gathered this type of data for years (e.g., [13]) and numerous research efforts have shown how these data can feed into usability practices (e.g., see [7] and [8] for surveys of techniques). However, the open source community lacks any dedicated infrastructure to collect this type of data, making it difficult for projects to empirically describe their user base.

In this paper, we introduce the concept of *open instrumentation* as a means for open source projects to collect data summarizing the actual, real-world practices of its users. Like other software instrumentation, open instrumentation implies the augmentation of an application to collect data regarding its use, such as what commands are used. However, in contrast to previous instrumentation efforts, open instrumentation follows the ethos of the open source community's culture of practice and makes all collected data publicly available. This public availability is intended to make it possible for non-project members to make meaningful contributions to the usability process, similar in spirit to the development process itself.

We demonstrate open instrumentation in *ingimp*, a version of the GNU Image Manipulation Program (GIMP) modified to collect five types of usage data: The commands used, high-level user interface events, overall features of the user's documents, summaries of the user's general computing environment, and users' own descriptions of their planned tasks. All collected data are automatically sent to a central web server, http://www.ingimp.org, where they are made publicly available for anyone to download and analyze. In addition to the raw data, the ingimp website provides a number of statistical summaries of the community's use of the software (e.g., most frequently used commands).

This paper's primary contributions lie in conveying the lessons learned in transforming the known technique of instrumentation to operate within the sociocultural context of open source software development. For the purposes of this paper, our primary focus is on how ingimp's design addresses issues of privacy and motivating use.

The public availability of collected data raises obvious concerns regarding privacy and anonymity. In particular, usage data should be collected in a way that minimizes the risk that sensitive personal information is collected. As such, we developed a set of conventions to govern the design of ingimp's data collection methods. We describe these conventions and their application to instrumentation, whether open or closed.

ingimp also constitutes a third-party "fork," or derivative, of the official software distribution. As such, there is the need to *motivate* use since the forked version does not offer significant, additional, end-user functionality. To address this need, the ingimp website features *personal* and *personable statistics* to compel use. Specifically, the ingimp application provides a command to automatically connect users to the ingimp website in a way that allows them to view their own *personal* statistics alongside the community's statistics. For example, the user can view their most frequently used commands alongside the community's most frequently used commands.

In addition to these personalized statistics, the ingimp website dynamically generates an ingimp *persona* for each user (Figure 1). The ingimp persona is a basic information visualization that augments an avatar with items depicting the user's typical use. These personas make the data more *personable* by summarizing information in more familiar terms. The personas also allow the creation of a "group snapshot" of the entire community of users.

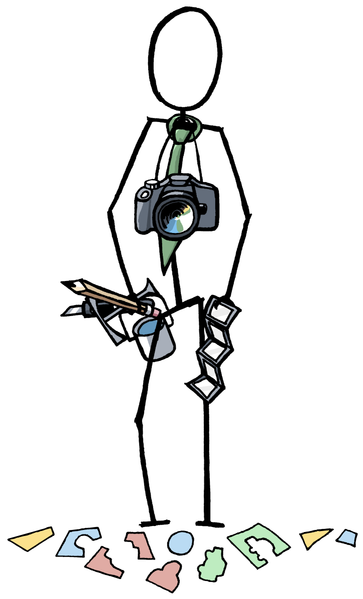


Figure 1. The ingimp website dynamically generates *personas* for each user (shown above). An ingimp persona summarizes the most frequently used commands (which are held in the right hand), the typical size of images worked on (held in the left hand), and the types of tasks (indicated here by the camera and clippings of images on the ground).

The rest of this paper is organized as follows. First, we review the concepts of open source software (OSS) and OSS development. We summarize current usability practices for OSS projects, which serves to motivate the need for usage data within this community. We then present the concept of open instrumentation and describe its instantiation in the design of ingimp, including mechanisms designed to motivate use. Next, we more fully describe the data collected by ingimp and the set of conventions we developed to guide decisions regarding what data to collect. Results from an initial qualitative study are presented and we conclude with directions for future work.

OPEN SOURCE SOFTWARE DEVELOPMENT

In this section, we provide a brief background on the concepts of open source software, starting with a discussion of its licensing model and associated development practices. We then review current usability practices in this community and identify unmet needs.

Open Source Software Licenses

Open source software, by definition, refers to software that includes access to the software's source code. Numerous philosophies and licenses intersect with this basic concept (e.g., free software [4], open source software [18], the GNU Public License [4], the BSD license [18]), but all include the basic provision of providing access to the software's underlying source code.

To be considered an open source license as approved by the Open Source Initiative (OSI) (a non-profit organization dedicated to open source software development [18]), an open source software license must meet a number of additional requirements. Among other requirements, an OSI-approved license must include the right for third parties to modify the code and redistribute derived works under the same terms as the original software. In practice, this provision is typically embodied in such a way that a third party can, at any time, create and distribute a derivative of the original software, without obtaining explicit permission to do so. This aspect of open source software is immensely powerful: As long as one follows the requirements of the original license, one can create a new application derived from existing open source software, saving significant time and effort compared to building the software from scratch. There are countless examples of third parties taking advantage of this property of open source software: Apple based their Safari web browser on the KDE project's HTML rendering code [23], Tivo based its digital video recorder system on a modified version of Linux, and the web browser Firefox forked from the Mozilla web browser.

The provision to allow derivative works can lead to a problem fairly unique to open source software: *identity management*. In particular, derivative works, or "forks," require end users to educate themselves about the different versions available to understand which version best fits their needs. While a similar problem can arise in the distribution of commercial software (e.g., the multiple versions of the Microsoft Windows operating system), open source projects have little control over who creates derivatives or how they are presented to the public. This property of open source licenses has implications for research work done within this domain since it requires research efforts to differentiate themselves from the official software efforts.

Open Source Software Development Practices

Open source software, in its strictest sense, refers to a particular licensing model for a piece of software. However, it is also strongly associated with a particular style of software development. In particular, open source software development is equated with a transparent, open process in which anyone can participate and make contributions [19]. Proponents of open source software maintain that this culture of practice confers a number of advantages, such as the rapid discovery and resolution of critical bugs [19].

A range of development tools help to create and maintain this open development process. Source code repositories (e.g., CVS or Subversion) coordinate activities from multiple developers; bug tracking software such as Bugzilla or Trac offer a dedicated forum for collecting, assigning, and discussing software deficiencies; and "diff" and "patch" allow individuals to submit source code modifications via mailing lists, making the barriers to contributing code extremely low. After an initial setup cost, these tools, for the most part, require little maintenance. They also serve to enforce particular work styles, which is important when managing the project. For example, Bugzilla provides a very structured way to submit, assign, and discuss bugs. This structure standardizes the bug reporting process and creates a certain consistency across *all* projects that use this infrastructure.

The features of current open source development tools and infrastructure collectively suggest a set of ideal characteristics to strive for in the development of any open source process support tool. In particular, any such tool should:

- Natively support the creation and manipulation of the data of concern (where in the case of software, the primary data is source code, represented by text)
- Make work practices open and transparent to the public to make public participation possible
- Provide a low barrier for public participation
- Scaffold desirable work practices
- Have low, long-term maintenance costs

While it may not be possible to embody all of these ideals, the more fully each is realized in a tool, the more likely it can leverage the unique attributes of the open source community's culture of practice. These desiderata are important to keep in mind when designing tools to support usability practices within the open source community.

Open Source Usability

The open source software community has demonstrated its capability to create systems software such as web servers (e.g., Apache) and operating systems (e.g., Linux and the various flavors of BSD). Having achieved this success, the community has identified software usability as an issue that needs to be better and more consistently addressed within its current development practices (e.g., [1, 14]).

"Usability" is a broad term that can refer to a wide range of software attributes, including efficiency, learnability, and subjective satisfaction [17]. The open source community is gradually starting to realize that achieving these goals requires a range of practices, including requirements gathering, design, and evaluation.

Current open source usability efforts are typically supported via general-purpose communication tools, repurposed development infrastructure, and usability-specific resources. We review each of these support mechanisms in turn and discuss their respective benefits and drawbacks.

General purpose communication tools, such as email, IRC, newsgroups, and blogs, have all been put into service to support usability efforts within the open source community [15, 22]. These tools support general, unstructured *text-based* discussions about usability issues. The medium of text ensures that *any* user can access and participate in these communications, but has obvious limitations in the realm of usability. In particular, other media can more effectively communicate visual designs, prototypes, and results from usability sessions. To partially address the deficiencies of text, blogs have been identified as a medium to support the presentation and discussion of prototypes [15]. However, blogs still lack tools for the *direct* production and manipulation of graphical data. As a consequence, the easiest way to build on, critique, or respond to these designs is by leaving a text-based comment on the blog.

Some of the existing development infrastructure has been repurposed to support usability efforts. Most visible is the use of Bugzilla to support the submission and tracking of usability issues. Bugzilla provides structure to help log and track usability issues, but it is primarily text-based (though one can attach images to bug reports). This reliance on text carries with it the same issues for the text-based communication tools described above.

In addition to these repurposed tools, a number of resources have been developed to specifically address usability concerns. Organizations such as Sun and Novell have conducted usability tests (e.g., [1, 2]) and contributed resources to support the construction of human interface guidelines (e.g., the GNOME Human Interface Guideline [5]). In the spirit of open source development, the results of lab-based usability tests, including the raw data itself, are publicly available at sites such as www.betterdesktop.org. Lessons learned from the usability tests have contributed to important redesigns of software such as the GNOME window manager. Similarly, the human interface guidelines have been shown to be a useful arbitrator when exploring design alternatives, suggesting their long-term utility [15].

Various grass-root efforts have also arisen to support open source usability. Examples include openusability.org and flossusability.org. openusability.org is a web-based service designed to pair usability experts with open source projects, to host usability discussions, and to serve as a repository for relevant usability information. At the time of this writing, the site no longer pairs experts with projects due to an apparent lack of usability experts. flossusability.org, on the other hand, organizes FLOSS (Free / Libre / Open Source Software) "sprints" to educate open source project members about usability techniques and practices.

Unmet Usability Needs in Open Source: Usage Data

The current set of practices adopted by the open source community has done much to begin to address usability issues in the development of open source software. But while projects are becoming more adept at discovering and discussing usability issues, they still lack data describing their user base: Who uses the software, for what purposes, how often, with what level of expertise, using what types of computing environments, etc. Benson, speaking from the position of a contributor to open source usability efforts, cites this deficiency, questioning how reliable it is to characterize a community of users by only considering the opinions represented on a project's mailing list [1]. Mailing lists and bug tracking systems provide a valuable means for users to identify unmet needs or usability issues, but research indicates that only a very small percentage of users actually participate in such forums [16]. Without these data, it can be difficult to prioritize efforts, because it is not generally known what aspects of the software are most important to the community in day-to-day use. The need for usage data is especially great for applications with a broad feature set, such as office applications or graphics applications, since these applications can be applied to a wide variety of tasks. In the absence of empirical data, developers must rely on instinct and anecdote when estimating what development efforts will have the most significant impact on the community of users. Software instrumentation is one means of obtaining data to answer these questions.

Software Instrumentation

Commercial software companies and researchers regularly deploy instrumented applications to understand the actual practices of users. These instrumented applications typically capture user interface events, such as command invocations, window events, and interaction with controls. For example, Microsoft Office includes facilities to track various features of how it is used [13]. Instrumentation is also commonly used to study use of web-based applications.

The data that results from instrumentation is most accurately characterized as *usage data*, as opposed to *usability data*. The distinction is subtle, but important, because it suggests what questions each data type can best answer. Usability data describes areas of an application's design that could be improved, and is the result of expert evaluations, heuristic evaluations, *in situ* observations, and other evaluation methods.

Usage data, on the other hand, is a rawer form of data. Usage data summarizes *how* the software is used, without attempting to interpret whether that usage indicates usability flaws or not. While past research has demonstrated how usage data can lead to the discovery of specific usability problems (e.g., see [7, 8]), we note that usage data, on its own, is useful by virtue of its ability to describe how a community actually uses the software on a day-to-day basis. For example, usage data can describe users' computing environments, the commands they typically use, and the types of documents they work on. These data, in turn, can feed into the design process by suggesting whether particular designs are likely to positively affect a significant number of users, given the community's computing environment and practices.

OPEN INSTRUMENTATION

Open instrumentation transforms the concept of instrumentation to match the ethos of the open source community: Both the instrumentation itself and the collected data are made available to the public. While simply stated, there are issues that must be recognized when publicly collecting and disseminating application usage data.

Open instrumentation is intended to address the OSS community's need to understand its user base, in a way that matches the community's culture of practice. However, the open nature of such instrumentation amplifies issues present in *any* software instrumentation. Specifically, there is the need to *minimize* the risk that sensitive, personal information could be collected, since all collected data are made public. Additionally, there is the need to *compel use* of the software, since the benefits of using the instrumented version are not immediate or tangible. We explore these issues later in the design of ingimp.

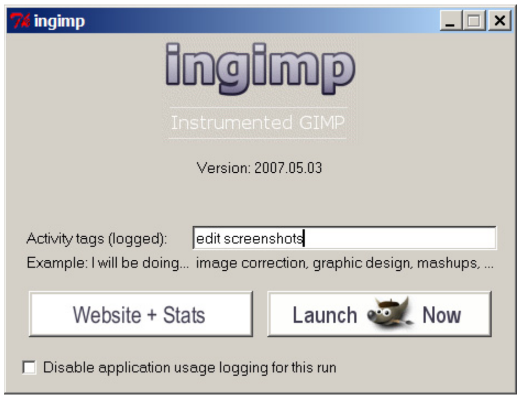
The notion of open instrumentation is not wholly new. Fedora's smolt [20] and Debian's popularity contest [3] collect data about the user's computer hardware, and what software packages are installed, respectively. Crash reports also represent a form of instrumentation, since they provide information about the state of the computer when it crashed [21]. Past research has instrumented open source applications to statistically determine where there are bugs in the software [10]. However, none of these efforts have collected detailed, high-level application usage data. Finally, recent research has demonstrated how a community can collectively report and avoid bugs while using the software [12]. We turn now to a description of ingimp, an openly instrumented application designed to collect this type of data.

INGIMP: END-USER DESIGN

System Overview

ingimp is a fork of the GNU Image Manipulation Program (GIMP), a general purpose bitmap editing application. ingimp adds instrumentation capabilities to GIMP to collect information about how the software is used in practice. All collected data are automatically transmitted to the ingimp website (http://www.ingimp.org). The website makes the data accessible in both raw and summarized form.

The majority of data collection in ingimp happens transparently in the background as the individual uses the software. However, ingimp also provides the option for users to describe, in their own words, what their intended tasks are. At start-up, users can indicate their intended task


ingimp

Instrumented GIMP

Version: 2007.05.03

Activity tags (logged): edit screenshots

Example: I will be doing... image correction, graphic design, mashups, ...

☐ Disable application usage logging for this run

Figure 2. The ingimp start-up screen allows users to describe their intended task, visit the website to view personal statistics, or launch GIMP. Users also have the option of disabling logging each time the application is started.

via "Activity Tags" (Figure 2). This free-form text field provides a means to discover how individual users perceive their tasks, in terms that are meaningful to them.

The ingimp start-up screen also includes a button labeled "Website + Stats." This button provides a direct link to the ingimp website and the statistics that have been collected; when pressed, it will cause a web browser to load the ingimp statistics page.

This direct link to the website is also the means by which the user can view *personal* statistics that summarize their own use. For example, users can see the most popular commands used by the community, as well as the most popular commands they, themselves, use.

ingimp User Experience: Client Application

ingimp augments the end-user's experience of GIMP with the following, additional elements: a consent agreement, the ingimp start-up screen, the ability to communicate post-installation updates and news, and a remote kill switch.

Consent Agreement

Before logging can occur, ingimp displays a consent agreement to which the user must consent before logging occurs. For reasons described below, users can deny consent, but still launch the application. In this case, the application will not collect data. However, each time the application starts, the consent agreement will be shown until consent is granted.

The somewhat unconventional interaction sequence described above (i.e., still launching the application, even if consent is not given) illustrates one of the subtle ways in which the open source ethos can affect interaction design. Specifically, in most cases, one would simply design the application to quit if consent was not given. However, doing so would violate some important tenets of some open source groups. In particular, to be included with the Debian Linux distribution, one must conform to their guidelines, which expressly forbid the inclusion of software that attempts to limit *who* can use the software. As such, to be included in this distribution, we needed to provide a means for users to continue to use the software, even if they did not consent to having their activity logged. This gave rise to the consent process described above.

The consent agreement provides details of the types of data collected and also describes ways a user's privacy could potentially be affected. Importantly, the consent agreement makes *no claims of confidentiality*. While we have designed ingimp to collect data that respects an individual's privacy, the public availability of the collected data makes it unreasonable to make any claims or guarantees regarding an individual's confidentiality.

ingimp Start Screen

Every time ingimp starts up, the start screen is shown (Figure 2). The start screen serves a number of purposes: It provides the option to disable logging, it provides facilities for users to enter "Activity Tags," and it offers a direct link to the ingimp website and its statistics. Most importantly, the start screen serves as a constant reminder that the user is using a specially modified version of GIMP. Accordingly, there is no option to disable the start screen (as is common with other informational start-up screens, like those that provide tips or task "wizards").

The Activity Tags' free-form text-entry field is provided to allow users to describe, in their own words, how they plan on using the software. Adopting common nomenclature, we call these descriptions "tags" to suggest one should enter pithy descriptions. Even with this suggestion, the potential variation in task description can be great. To help constrain the types of responses, the design includes two elements to suggest how to fill out the field. First, we adopt the convention of completing a sentence: A label to the left of the entry field reads, "I will be doing…" suggesting that the sentence should be completed. Second, below the input field, an example illustrates one way to complete the sentence.

All text entered in the Activity Tags field is added to the log, without alteration. To make this point clear, and to remind users of this fact each time they enter text, the label for Activity Tags includes the parenthetical note "(Logged)".

The ingimp start screen includes two push buttons, one that provides access to the website, and a second that launches the application itself. The "Website + Stats" button serves several purposes. When clicked, it opens a local browser window to the ingimp statistics page, reducing the need for the user to know where or how to view data collected from the application. To the best of our knowledge, this represents the first example of an instrumented application providing a direct link to the collected data.

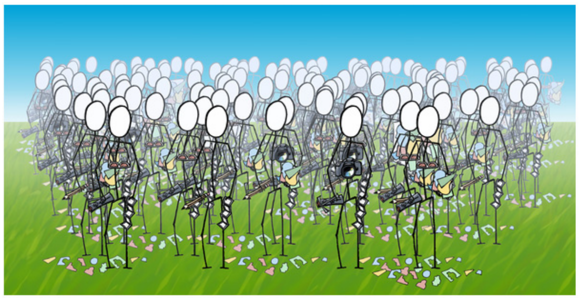


Figure 3. A "group shot" of the ingimp community, derived from individual ingimp personas. This shot is dynamically generated and displayed on the ingimp website.

The "Website + Stats" button also serves as the means by which users can view their own, personalized statistics. If the user were to go directly to the ingimp website, it would be difficult to provide a method for users to selectively display their own personal statistics, since the gathering of the statistics is meant to provide a measure of anonymity. To get around this problem, the "Website + Stats" button sends the user's randomly generated ingimp ID when requesting the page. This idea serves to filter the data presented to the user.

Post-Installation Feedback

When quitting the application, the logged data is automatically transmitted to the web server. During this transmission, the web server can optionally send back a text-based message to display to the user. This feedback mechanism is similar to other informational, update mechanisms commonly found in software, though it does not currently provide the means for automatically updating the software. Instead, it is intended to provide the ability to directly communicate any new study information to the user.

Kill Switch

ingimp provides a "kill switch" to allow researchers to remotely disable ingimp's logging facilities. Each time the application is closed, the software checks for this remote kill switch. The remote kill switch provides a means for researchers to inform subjects that the study is done, and to disable the experimental software. (The software will continue to function, minus the logging features.)

ingimp Website: Personal and Personable Statistics

Past research suggests that users value feedback when participating in remote usability efforts [6]. As such, ingimp provides such feedback through the *personal* and *personable statistics* available on the website.

The aforementioned "Website + Stats" button takes users directly to the ingimp *persona* page. For each user, a persona is dynamically generated that summarizes their typical usage (Figure 1). The persona is a basic information visualization: Representations of the most frequently used classes of tools are held in the persona's right hand, while the left hand holds a canvas related to the user's typical tasks and common image sizes. For example, if the user normally works on relatively large images (of several mega-pixels), the persona is shown holding a large canvas. Conversely, if the user typically works on relatively small images (for example, for web pages or icons), the persona is shown holding a set of small, wallet-sized images.

Each user is assigned to one of five different "use" categories: new users, those who manipulate photographic images, graphic designers, painterly artists, or "cut and paste" artists (e.g., those who create "mash-ups"). These classifications arose from an initial study of users, but should not be interpreted as a canonical set of types of users for ingimp. Rather, they serve as a useful starting point for making expressive personas. The generation of the personas is driven by a combination of hand-coded rules and k-means clustering. A more complete discussion of the mechanisms for deriving personas is beyond the scope of this paper.

The ability to create personas for each individual enables the creation of a "group snapshot" of the entire community, which is shown on the website (Figure 3). At present, personas are randomly arranged within the group snapshot; there are obvious opportunities to enhance the composition of the group shot to create other information visualizations. For the time being, the group snapshot serves to give users a sense of community, and a sense of belonging to a larger effort.

Given this view of ingimp from the user's perspective, we turn now to a description of its internal data collection methods.

INGIMP: DATA COLLECTION AND PRIVACY

ingimp collects five primary types of data:

1. Commands used
2. Fundamental user interface events
3. Features of the user's documents
4. Information regarding the user's general computing environment
5. User's own (optional) descriptions of their tasks

The choice of data types was driven by an examination of the types of data collected in previous instrumentation research (e.g., [7, 8]) as well as results from our own observational studies of users that suggested key types of data to collect. This initial set of data closely mirrors past instrumentation work, while adding some relatively unique measures (such as features of users' documents). We briefly describe each type of data, followed by a discussion of privacy issues associated with these data. However, in this paper, we do not describe how we are making use of these data. Instead, we refer interested readers to the ingimp website to view our current statistical analyses.

Commands

ingimp automatically logs all commands that appear on the undo stack. Command names, but *not* command parameters, are directly recorded in the log file. While command parameters would provide additional data useful for understanding users' tasks, this data would enable reconstruction of a user's personal work process.

The collection of command names helps indicate the types of tasks in which people engage, the variability of their tasks, and their potential level of expertise (i.e., some commands are indicative of more expert use).

While ingimp does not collect parameters for commands, it does record a summarization of the strings used for file names and layer names. These summaries help track activity across sessions (e.g., whether they repeatedly work on the same file) and whether users customize layer names. Specifically, ingimp records the number of letters, numbers, punctuation marks, and forward and backward slashes in strings. Additionally, ingimp generates and records a 32-bit hash of the entire string to help track string usage across sessions. In the original release of the software, this 32-bit hash was directly recorded in the log file. However, this 32-bit number, together with the length of the string, allows one to perform a brute-force search to discover the original string. Accordingly, the current version of ingimp generates an arbitrary 32-bit number to associate with the generated hash. The association between the arbitrary number and the hash is recorded on the user's machine, but only the arbitrary number is recorded in the log file. This method enables one to track the use of identical file names and layer names across sessions, while substantially decreasing the chance that a third party can reconstruct the original strings.

User Interface Events

ingimp logs fundamental user interface-level events: Window events (focus, move, resize), keyboard and mouse usage (but *not* the actual keys or mouse locations), the state of modifier keys, menu usage, and tool selections. Mouse button events are recorded and include the button pressed, *but not the location of the cursor*. Cursor location would allow reconstruction of a user's work with tools such as the paintbrush, and thus impact users' privacy. Tool selection changes are recorded (e.g., selecting the paintbrush), though any parameters specific to the tools are not recorded (such as the brush size or its color). These interface events help to characterize interaction preferences, such as document window sizes, the preference for the mouse versus keyboard, and so on.

Document Characteristics

Unique to ingimp is the collection of data that characterizes the documents themselves. In particular, ingimp collects information regarding:

- The number of layers in an image
- Image and layer sizes
- Histograms of images and their individual layers

The use of layers suggests a certain degree of sophistication with the application. That is, novices typically do not use layers, being unaware of their presence or functionality. As such, this information helps to categorize the various types of users.

The image and layer sizes also help to characterize the types of data users work on. For example, standard digital camera image sizes can be easily identified.

Image histograms are constructed from the pixel values and provide a richer description of the types of images that users operate on, without revealing the actual content. One of the most immediate uses of these histograms is to distinguish between photographic images and other types of graphics, such as line art; the former are typified by full, variable histograms, while histograms for the latter tend to have very "spiked," sparse peaks because of the low number of colors in the image.

Computing Environment

ingimp records basic characteristics of the user's computing environment: Their operating system, the number of monitors, and the resolution of each monitor. ingimp also records the time zone of the user, but not their precise geographic location. Though the user's location could be approximated via their IP address, we do not make use of their IP address for privacy reasons. These data suggest the general geographic regions of ingimp users.

Activity Tags

As described above in the "User Experience" section, users can use Activity Tags to enter free-form text-based descriptions of how they plan on using ingimp.

Addressing Potential Privacy Concerns

Making collected usage data publicly available creates obvious privacy concerns: There is always the possibility that personal information could unintentionally be discovered about the user, even if that information was not explicitly collected. This risk is present in *any* software instrumentation, whether closed or open, but the issue is particularly salient in open instrumentation.

Recognizing that it is impossible to guarantee complete anonymity in the design of any instrumented software application, there were a number of conventions we employed to help minimize privacy concerns. These can be summarized as follows:

- Do not collect command parameters, just command names
- For user-supplied strings that are directly recorded in the log file (i.e., strings supplied by the user), make it clear to the user that the string will be directly recorded in the log file without alteration
- Record summarizations of strings if necessary, but use arbitrary keys to track strings across sessions
- Know what data the functions will log
- Record the log file in a human-readable format

Do Not Record Parameters

The first guideline, do not record parameter values, helps to prevent third parties from recreating the work of another user. In the design of ingimp, we found a number of non-obvious applications of this guideline that illustrate its utility. As an example, when the user selects a color for the paint brush, one could assume that this single piece of information (brush color) is innocuous enough to record. However, recording all colors chosen by a user would allow a third party to recreate a user's color palette, which could be considered part of a graphic designer's "signature look." Since paint color constitutes a parameter for a command (where the command is to paint), application of this rule helps avoid this issue.

Alert Users to Directly Recorded Strings

Strings represent some of the most personal pieces of information that could be recorded in an instrumented application. As such, we are careful to highlight cases in which strings are directly copied into the log file. Application of this rule can be seen in the Activity Tags reminder that the tags are logged, and in the consent agreement, which indicates command names are directly recorded.

Summarize Other Strings When Necessary

If it is useful to note features of a string (e.g., for the purposes of understanding file naming conventions, or how often users stray from automatically generated names, such as "Layer 1," "Layer 2," etc.), strings can be summarized in the log file, rather than directly copied. As noted above, however, the tracking of strings (or other objects) across sessions should be done using an arbitrary ID.

Know What Data Is Recorded

This guideline seems obvious and not necessary to state. However, it is important to consider for the simple reason that one must understand the different circumstances under which a logging routine could be called, and with what types of data. To illustrate this point, consider the logging of command names in ingimp.

GIMP has a plug-in architecture that allows third parties to extend its capability with scripts and plug-ins. Each script or plug-in has its own unique name, supplied by the author of the extension. As such, a user could create a personal script and use it in GIMP. However, if the script has personal information in the script name (e.g., "ACME Widgets' Rotoscoping Script"), this information will be directly recorded in a log file as-is. Because of this possibility, ingimp explicitly describes this scenario in its consent agreement to alert users to this potential privacy problem.

Use Human Readable Log Formats
ingimp's file format is XML-based. It uses intentionally verbose, descriptive names to increase the ability to comprehend the log files without the need for additional tools or instruction. Raw, unprocessed log files are available on the website to satisfy curiosities or questions users may have about what is logged.

The guidelines presented above should not be taken as a canonical (or sufficient) set of rules for addressing privacy concerns in openly instrumented application. However, as we designed the application and developed these heuristics, we found them useful in guiding decisions regarding what data to collect and not collect. Future efforts should continue to explore and evaluate data collection policies for openly instrumented applications to understand users' perceptions and concerns, as well as implications for what can or cannot be known about users via the collected data.

INGIMP: INITIAL EVALUATION
ingimp, both the client and website, was evaluated using interviews and a think-aloud observational study. The intent of these sessions was to holistically evaluate the entire ingimp application, both the client and website, to understand how people used and reacted to the open instrumentation and its features.

Six subjects participated in this initial study. All were undergraduate university students. Using a machine we supplied, each user was asked to navigate to the website, locate the software, and download and install it. They were then asked to create a logo for the software. When finished, we asked subjects to view their statistics on the website. Sessions lasted 45-60 minutes each.

Summary of Findings
Most users in our study spent little time reading the software's consent agreement, though some did take the time to carefully read it. This discovery suggests the need to enhance the likelihood that end-users read and comprehend the consent agreement so they fully understand the risks involved in using ingimp. We have begun work addressing this issue by including illustrations with the consent agreement text that visually depict ingimp's data collection.

After completing the task, we asked subjects to examine their statistics on the website. This request revealed that the majority of users did not notice the "Website + Stats" button when they first started the software, suggesting the need to explore additional ways to bring this functionality to users' attention. This remains an open problem.

Once subjects did navigate to the website, they all enjoyed exploring the statistics. Our subjects particularly liked to browse the statistics summarizing command use, often commenting that it would be a useful mechanism to discover functionality in the software that they were not aware of. This finding suggests that the data collected could also serve as a tool to help users become more adept at using the software, similar in spirit to related efforts that have used usage data for this purpose [11].

The ingimp personas were also favorably received. A number of suggestions arose concerning their design and presentation. First, some subjects expressed the desire to see how their persona changes over time. Such a feature could help indicate whether one's expertise is maturing. Users also indicated that it would be useful to provide a key to describe the various elements in the visualization. Such a key could be quite literal (akin to legends on maps), or interactive. For example, one user suggested the ability to mouse over portions of the visualization to get more detail about that component, including the numerical data represented by the component.

When interviewed about privacy concerns, subjects did not express any concerns about ingimp's logging capabilities. One student bluntly stated, "We are students, we don't care about privacy." Apart from this extreme response, students cited a handful of reasons that privacy was not a large concern: The software is an image manipulation application, and thus does not handle sensitive, "private" information; the source code is available, leading them to assume that the community could "police" the software and discover any privacy issues; and there was a general degree of trust that the software would not attempt to maliciously collect information. In some sense, these reactions bode well for future open instrumentation efforts. On the other hand, they suggest that the average user may not fully understand all of the potential implications of open instrumentation. It would be useful to more widely explore views of open instrumentation in the open source community.

The results of this initial study suggest that open instrumentation is an option worthy of further exploration. In particular, the statistics, both for individuals and the community, seem to be compelling to users. The dynamically generated personas also seem to favorably predispose people to the notion of instrumentation. Though demonstrated in the context of an open source application, it is likely that these techniques would be favorably received in other contexts, as well.

CONCLUSION AND FUTURE WORK
Open source software has matured from software made for and used by hobbyists, to software that industry, governments, students, and many others rely on, on a day-to-day basis. It has been proven to be a viable, inexpensive alternative to commercial offerings, and is increasingly advocated for use in developing countries' IT infrastructure because of the benefits it can accrue to local economies [24]. The usability of open source software can thus have a profound effect on the hundreds of thousands of people

who adopt it for philosophical, political, and/or economic reasons, making it in an important issue for the HCI community.

This paper has presented work aimed at addressing one identified need of existing open source usability efforts, namely, the collection of usage data. We presented methods to motivate end-user use of instrumented software and described a set of conventions to guide the design of data collection to help minimize privacy concerns. The lessons learned in the design of the client and website are directly applicable to other instrumentation efforts, though we caution other open source projects to proceed carefully with similar efforts. Open instrumentation is a delicate issue, the consequences of which we are still actively exploring. While ingimp demonstrates that it is possible, the long-term benefits must clearly outweigh any potential impact to users.

ingimp was first deployed in May 2007 and has been installed by over 700 users in the first six months of its release. We are now just beginning to analyze the data being collected, but the data analysis has demonstrated one particular need for future work. The ingimp site currently has a modest set of statistical summaries, but more are clearly needed. However, one of the current limitations of the current ingimp website is that the public cannot actively participate in the development and discussion of statistical analyses through the website itself. Thus, our next line of investigation is to examine the possibility of extending the website to allow users to post SQL-like code to create their own statistical summaries. These posts may follow a wiki or blog-like format, but the overall goal will be to move beyond the fixed summaries we provide, to truly open up the data analysis process to the entire community.

REFERENCES

1. Benson, C., Muller-Prove, M., and Mzourek, J. 2004. Professional usability in open source projects: GNOME, OpenOffice.org, NetBeans. In *CHI '04 Extended Abstracts*, pp. 1083-1084.
2. Better Desktop. http://www.betterdesktop.org
3. Debian Popularity Contest. http://popcon.debian.org
4. The Free Software Foundation. http://www.fsf.org
5. GNOME Human Interface Guidelines 2.0. http://developer.gnome.org/projects/gup/hig/2.0
6. Hartson, H. R., Castillo, J. C., Kelso, J., and Neale, W. C. 1996. Remote evaluation: the network as an extension of the usability laboratory. In *Proceedings of CHI'96,* pp. 228-235.
7. Hilbert, D. M. and Redmiles, D. F. 2000. Extracting usability information from user interface events. *ACM Comput. Surv.* 32, 4 (Dec. 2000), 384-421.
8. Ivory, M. Y. and Hearst, M. A. 2001. The state of the art in automating usability evaluation of user interfaces. *ACM Comput. Surv.* 33, 4 (Dec. 2001), 470-516.
9. KDE Human Interface Guidelines. http://usability.kde.org/hig
10. Liblit, B., Aiken, A., Zheng, A. X., and Jordan, M. I. 2003. Bug isolation via remote program sampling. In *Proceedings of the ACM SIGPLAN 2003*, pp. 141-154.
11. Linton, F., Schaefer, H.P.: Recommender Systems for Learning: Building User and Expert Models through Long-Term Observation of Application Use. *User Modeling and User-Adapted Interaction* 10(2-3): 181-208 (2000)
12. Michail, A. and Xie, T. 2005. Helping users avoid bugs in GUI applications. In *Proceedings of the 27th international Conference on Software Engineering* (2005). ICSE '05. ACM, New York, NY, 107-116.
13. Microsoft Office Customer Experience Improvement Program (CEIP). http://office.microsoft.com/en-us/help/HA100377971033.aspx
14. Nichols, D.M. & Twidale, M.B. (2003). The Usability of Open Source Software. *First Monday* 8(1) - January 6th, 2003.
15. Nichols, D.M. & Twidale, M.B. (2006). Usability Processes in Open Source Projects. *Software Process - Improvement and Practice Journal: Special Issue on Free/Open Source Software Processes*. 11(2) 149 – 162.
16. Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., and Ye, Y. 2002. Evolution patterns of open-source software systems and communities. In *Proceedings of the international Workshop on Principles of Software Evolution* (Orlando, Florida, May 19 - 20, 2002). IWPSE '02. ACM Press, New York, NY, 76-85.
17. Nielsen, J. Usability Engineering. Boston: Academic Press. 1993
18. Open Source Initiative. http://www.opensource.org
19. Raymond, E.S. The Cathedral and the Bazaar. *First Monday*, 3, 3 (March 1998), at http://firstmonday.org/issues/issue3_3/raymond
20. Smolt. https://hosted.fedoraproject.org/projects/smolt
21. Socorro Crash Reporter. http://crash-stats.mozilla.com/
22. Twidale, M.B. & Nichols, D.M. (2005). Exploring Usability Discussions in Open Source Development. *Proceedings*, *HICS'05*, track 7, p.198c.
23. WebKit. http://developer.apple.com/opensource/internet/webkit.html
24. Winschiers, H. and Paterson, B. 2004. Sustainable software development. In Proceedings of the 2004 Annual Research Conference of the South African institute of Computer Scientists and information Technologists on IT Research in Developing Countries, pp. 274-278.