Usability Themes in Open Source Software

Jim Hall
University of Minnesota

(Dr. Ann Hill Duin, advisor)

April 30, 2014

ABSTRACT

This research examines the prevalent state of usability in open source software, focusing on the reasons why usability is often overlooked in the open source software noosphere. A usability test of GNOME, a popular open source software desktop environment, provides insights into the present development structure, and highlights areas for improvement. Analysis of the test data suggests features or *themes* of usability, and provides avenues of exploration to improve overall usability within open source software systems.

> A program should follow the 'Law of Least Astonishment.' What is this law? It is simply that the program should always respond to the user in the way that astonishes him the least.
>
> *The Tao of Programming* (pp. 55-57)
> Geoffrey James

Open source software developers create an array of innovative programs: *WordPress is the world's most popular blogging platform, used by a staggering 202 million websites … Magento, used by 30,000 merchants, including Samsung, Nespresso and The North Face, is the world's fastest growing e-commerce platform … Firefox currently accounts for 24.43% of the recorded usage share of web browsers, but this figure is on the rise … GnuCash provides a great, free alternative to paid-for accounting software … Music software like Cubase and Logic Pro can be incredibly expensive, which is why an increasing number of people are turning to Audacity, a free, cross-platform sound editor … Just like sound editors, industry standard image editing software is prohibitively expensive for a lot of people, but GIMP provides a free alternative … With the ability to create text documents, spreadsheets, presentations and databases, OpenOffice is an accomplished rival to Microsoft Office … 7-Zip is an extremely popular file archiver for Windows, which, although free, outperforms Winzip … Blender is a 3D content creation suite which can be used for everything from modeling to skinning, particle simulation, animating and texturing … Ubuntu is a free operating system for Linux that's quick and easy to use* (Walker, 2010). But open source software is often plagued with poor usability. End users of many popular open source software programs share comments about those packages that are both praise and criticism: "___ is a great program … once you figure out how to use it" or "You'll like using ___ … after you learn the awkward menus." These statements exemplify both gratitude for and frustration with open source software. Lack of strong usability hurts the user, damages the reputation of developers, and increases distrust of open source

software. Developers need to stop doing only what is comfortable for them and take steps to benefit the user through usability testing.

Usability is often ignored in open source software development. This is not specific to open source software; many software programs suffer from poor usability (Redish, interview, 2012). In this respect, usability in open source software is no different than in proprietary software. At best, open source software has the same usability problems as proprietary programs. At worst, open source software has worse usability in general because most open source developers tend to focus on the functionality and the features, and do not give much thought to the user interface. Although some open source software projects may have a maintainer who dictates a particular design aesthetic, open source software advocate and Open Source Initiative co-founder Eric S. Raymond comments that most programmers view menus and icons "like the frosting on a cake after you've baked it" (Raymond, interview, 2012).

Described in fundamental terminology, open source software is any program that permits users to examine the source code, the base instructions that govern the software's operation.[1] Via the collaborative medium of the Internet, developers work together to create new projects and solve problems too difficult for a single developer to accomplish. Describing this synergetic methodology and its ability to overcome substantial obstacles, Raymond shares this witticism: "given enough eyeballs, all bugs are shallow" (Raymond, *Bazaar*, 2000).

[1] Although Free Software Foundation founder Richard M. Stallman disagrees with the "Open Source Software Definition" (OSI, v. 1.9) and "open source" terminology in favor of his own "Free Software Definition" (Stallman, v. 1.111) and "Free software" taxonomy (Stallman, "Why Open Source misses…"), there is no significant practical difference between them. The essence of both terms is that anyone may view and modify the program's source code, and that both the program and its source code may be shared with others. This article assumes the more general definition of "open source software."

Raymond draws a comparison between building *structures* and building *software*. In traditional, proprietary software development, developers work in "splendid isolation, with no beta to be released before its time" (ibid). According to Raymond, this development model isolates the developers from the users, similar to the way Raymond envisions craftsmen laboring in seclusion to build a cathedral, disconnected from the community the structure is meant to serve. In contrast, Raymond characterizes the open source software development model as one where developers work in partnership with users, creating software that initially represents a solution to a particular urgency, but is then generalized and expanded to suit a wide variety of similar requirements. Different developers might work independently on separate programs, and later merge their efforts or intermix proposed solutions by combining source code implementations. To Raymond, this model parallels "a great babbling bazaar of differing agendas and approaches … out of which a coherent and stable system could seemingly emerge only by a succession of miracles" (ibid).

The "bazaar" provides a concise description of the open source software development model, visualizing a meritocracy where everyone may contribute, but only the best ideas are promulgated and get merged into the source code. Over time, such iterative improvement results in rapid advancement. Successful open source development requires that developers "release early and often, delegate everything you can, [and] be open to the point of promiscuity" (ibid).

According to Raymond, open source software developers tend to prefer assembling the ingredients and baking the cake over applying frosting to make it look nice (Raymond, interview). Developers see the user interface merely as uninteresting window dressing. This view is widely shared within the open source software community and is the fundamental misunderstanding between developers and users. Neglecting usability reflects a general cultural

trend within open source software: a dichotomy between developers and users, between those who enjoy the challenge of making new things work and those who want it to be attractive and simple to use. Usability is overlooked or ignored at the peril of open source software. If software is not attractive or easy to use, people will not want to use it. To resolve this dilemma, open source software developers must adopt usability testing to improve their product, enabling their audience to accomplish tasks quickly and easily.

I. What is usability

In *A Practical Guide to Usability Testing*, Joseph Dumas and Janice Redish provide a constructive definition of usability: "Usability means that the *people who use the product* can do so *quickly and easily* to accomplish *their own tasks*" (1994, p. 4). Dumas and Redish's definition rests on four components: (p. 4)

1. Usability focuses on users.
2. People use products to be productive.
3. Users are busy people trying to accomplish tasks.
4. Users decide when a product is easy to use.

In a more practicable view, average users with typical knowledge should be able to utilize the software package to perform tasks. The purpose of *usability testing*, therefore, is to uncover issues that prevent general users from employing the software successfully. As such, usability testing differs from quality assurance testing or unit testing, the purpose of which is to uncover errors in the program. Usability testing is not a functional evaluation of the program's features, but rather a practical determination of the program's operability.

Usability testing does not rely on a single method. There are multiple approaches to implement usability practices. While Dumas and Redish present a structured approach to usability testing, Preston (2004) describes eleven different techniques to evaluate usability, representing a spectrum of methods from interviews and focus groups to formal usability testing:

1. Interviews and Observations

Interview individual users about how they use a product to accomplish tasks. Later, observe the users as they use the product and document how they actually interact with the product.

2. Focus Groups

Interview a group of users about a product. Typically done well before the product exists, the focus group interview often asks about new features that might be useful.

3. Group Review or Walk-Through

Present a possible design to a group of users, who comment on it.

4. Heuristic Review

A usability expert evaluates an existing design and provides comments about how users are likely to respond to the product.

5. Walk-Around Review

Similar to the Group Review, post several versions of a design in different parts of a room, and ask a group of users to evaluate each design.

6. Do-it-Yourself Walk-Through

Simulate using the new design by walking through the design by yourself, using realistic scenarios and tasks. This is essentially a usability test without inviting any testers.

7. Paper Prototype Test

Print on paper all program menus and screens that a user might encounter when using the product. Simulate interaction with this paper prototype by presenting new menus and screens to a person who is testing the interface.

8. Prototype Test

Similar to the Paper Prototype, test an early version of the product using realistic scenarios and tasks.

9. Formal Usability Test

Invite a group of people to test a product, which might be the actual product, an early version of the product, or a paper prototype. Ask the participants to use the product with realistic scenarios and tasks, and observe how they interact with the product.

10. Controlled Experiment

> Invite a group of people to test two versions of a product or two different products, using realistic scenarios and tasks. Provide controls and statistical balancing to generate a detailed comparison of the two products.

11. Questionnaires

> Survey users about a product or a design.

Whatever method is employed, the value of usability testing lies in performing the evaluation *during* development, not after the program enters functional testing when user interface changes become more difficult to implement. In open source software development, the community of developers must apply usability testing iteratively throughout development. Developers do not require extensive usability experience to apply these usability practices in open source software. As suggested by Redish (email, 2012) developers can gain significant insight through observation, gathering a few test participants and watching them use the software. With each iteration, usability testing identifies a number of issues to resolve (ibid) and uncovers additional issues that, when addressed, will further improve the program's ease of use.

Usability cannot be addressed only at the end of a software development lifecycle. To be successful, usability evaluation must be part of the design and development methodology, and thereby addressed within the software development process. For too long, open source software has been developed *by* developers *for* other developers, focused on an inward audience and not attentive to an outward adoption. As a result, the open source software community now faces an exigence to incorporate usability practices into that methodology, or risk alienating end users through substandard usability.

II. Lack of usability testing is a problem

In open source software, developers contribute collaboratively to a computer program, then share the software for anyone to use for free. Users are therefore invited to become co-developers, modifying the program's source code to fix problems or extending the software to accomplish greater tasks. Nondiscriminatory access to the program's source code allows interested programmers to exchange ideas and information, which in turn enables rapid development of enhancements and improvements.

However, most open source software developers focus on contributing functionality or resolving errors in program behavior. Many open source software programs are written by developers in close association with other developers. Due to this close proximity, common values dominate, and projects may unintentionally target like-minded users: *i.e.*, other developers. Few contributors seek to address how *general* users with *average* knowledge might approach the software. Although some larger open source software projects such as Drupal make specific efforts to address usability issues (Drupal, 2011), most projects lack the resources or interest to do so. As a result, open source software programs are often utilitarian, focused on functionality and features, with little attention directed to user interaction.

The mounting challenge in open source software is to make programs more approachable by general users. Echoing Redish, people who use the product must be able to do so quickly and easily to accomplish their own tasks. This is the essential definition of *usability*, and represents a colossal shift in open source culture. To date, usability has been antithetical to open source software developers' preferred work model. Creating new features takes priority, and volunteer programmers rarely consider how new users will access that functionality.

But usability need not remain independent from the open source software development method. When users are invited to become co-developers, an opportunity exists to introduce usability. In open source software projects, the user community plays a strong part in testing each new release of the program. While this approach successfully uncovers errors in functionality, developers cannot rely on the established testing cycle to provide actionable usability feedback. Left on their own without an understanding of usability testing methodology, open source software testers respond with vague bug reports such as “This feature is confusing.” Such imprecise feedback is unhelpful in open source software development, as the feature may not be confusing to others, including the original developer, resulting in such bugs being categorized as “works for me” and thereafter ignored. In addition, bugs classified as “usability” may not be afforded the same status as functionality bugs, leading to an inherent developer bias against addressing usability issues (Nichols and Twidale).

Identifying usability issues in open source software requires a more structured approach. Open source software developers can apply a variety of methods, although the ideal is to conduct formal usability tests at every major release. Unfortunately, such formality is often antithetical to the open source software community preferred mode (Stallman, email, 2012). Convincing the open source community to adopt a widespread standardized usability practice is like “swimming against a strong cultural headwind” (Raymond, interview, 2012).

III. Demonstrating a usability test

Usability testing is most effectively employed during the development phase of software development. Open source software developers should employ usability testing throughout the development cycle. Usability testing during this phase is not difficult and yields useful results. One way to demonstrate the efficacy of usability practices is to conduct a usability test against an open source software project or set of open source programs. The results of such an examination would identify assets to and challenges in design concepts. In addition, the conclusions would inform future design and development across other open source software projects, working in a reciprocal relationship.

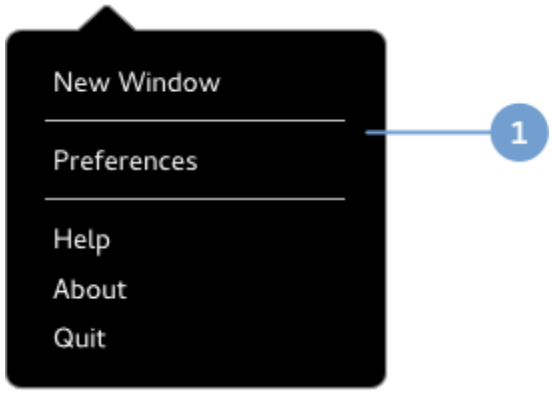
Demonstrating usability testing in open source software requires a project of appropriate scope. Choosing which open source programs to examine demands careful consideration. To generate useful results that may be generalized for other open source software projects, the programs of interest should not be too large, because varied and exhaustive scenarios confound the usability test analysis. Nor should the target programs be too limited, as trivial programs will not support generally supportable or applicable conclusions. Further, the program should be approachable by general users; programs that require specific knowledge present a restrained application of usability principles.[2] GNOME fits the test parameters in all respects.

[2] In a similar, previous usability test of open source software, commenters on several online forums proposed open source software programs that demonstrated good usability (Hall, "Candidates for good usability," 2012). Sorting through these suggestions, three open source software programs met the criteria for the usability test: the Firefox web browser, the Gedit text editor, and the Nautilus file manager. Seven testers participated in a usability study, using GNOME 3.4 on Fedora 17 Desktop Edition. The analysis of this test presented four themes to successful usability and recommended open source software projects incorporate usability testing into their development cycle (Hall, *Linux Journal*, 2013). This study expands on the previous research, focusing on the GNOME components.

GNOME is a desktop environment composed entirely of open source software, intended to be “an easy and elegant way to use your computer” (GNOME, website). Collaboration with the GNOME design team in the present usability test focused research on the question, “How well do users navigate and understand the design patterns in GNOME?”

The GNOME Human Interface Guidelines documents 14 design patterns. In this context, design patterns refers to general, reusable visual elements that make up a user interface, including these illustrated examples: (GNOME, HIG)

Application menus ① provide access to global, top level actions and options for your application. These include standard items for accessing documentation and information about the application.



A header bar ② is a horizontal toolbar that is located across the top of a window. Header bars contain commonly used controls which affect the content below. They also provide access to window controls, including the close window button and window menu.

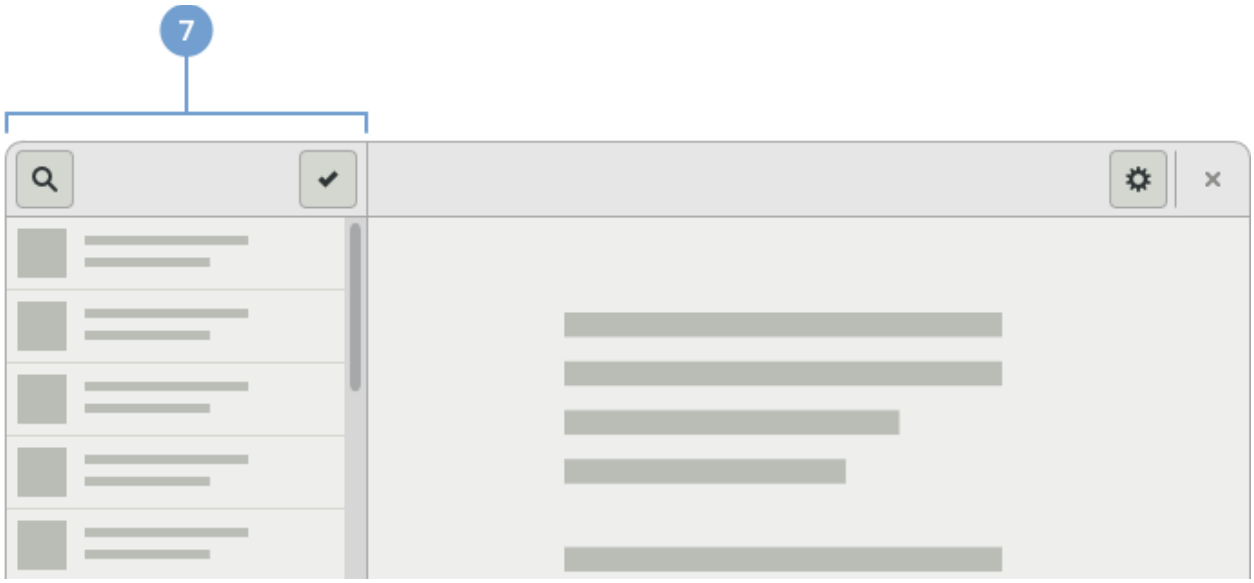
A view switcher ③ is a control that allows switching between a number of predefined views. It appears as a set of toggle buttons that are placed in the center of a header bar.

Search ④ can be a fast and effective way to find content, and consistently available search facilities are an important aspect of GNOME 3 application design. Search should be available whenever your application presents a large number of content items.

Selection mode ⑤ is used in conjunction with lists and grids. It allows actions to be performed on items of content. When selection mode is active, check boxes allow items to be selected. An action bar is shown at the bottom of the view, which contains the various actions that can be made on the selected content.

A gear menu ⑥ contains actions for the current view. It is accessed through a button that is placed at the right hand side of the header bar.

A sidebar list ⑦ is a pattern for structuring a whole application. It organizes the application into two panes of content, with a list on one side and an item of content on the other. The sidebar list effectively replaces the content overview pattern.



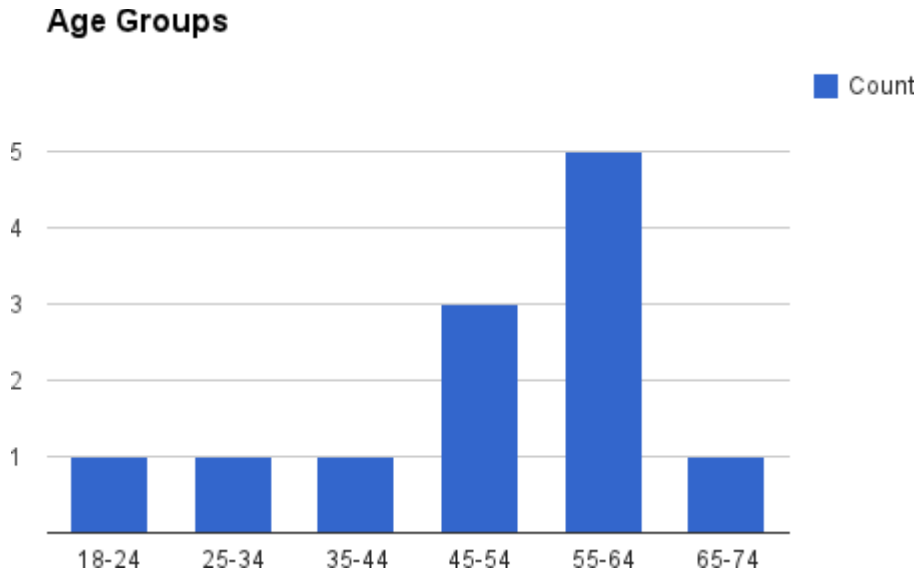
To support generating broadly applicable conclusions and themes, this usability test examines *design patterns* in GNOME, as demonstrated through several representative GNOME applications and programs.[3] The usability test examines these design patterns across multiple applications, allowing comparisons of use within different contexts. While this is not a rigorous statistical

[3] This is not the first time I have proposed applying usability evaluation to open source software. In 2012, I conducted a limited usability study of three common open source software programs: the Firefox web browser, GNOME's gedit text editor, and GNOME's Nautilus file manager. In analyzing this usability test, I identified four key themes that may guide successful usability in other open source software projects: Familiarity, Consistency, Menus, and Obviousness. In addition, I concluded that open source software contributors do not require particular expertise to effectively apply usability testing methodology to the open source software development lifecycle, and provided general recommendations to incorporate positive usability practices (Hall, *Linux Journal*, 2013).

analysis, it still falls within the requisite parameters described by Preston and supports generalizable conclusions.

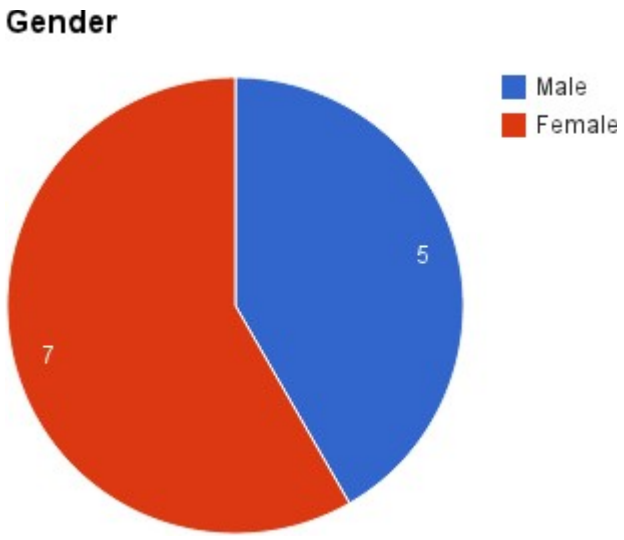
IV. Usability test design

Twelve testers participated in the usability test, representing a mix of genders and an age range spanning 18 to 74.[4] Participants self-identified their level of computer expertise between "2. I know some things, but not a lot" and "4. I am better than most," with an average rating of 3.25 and a mode of "3. I am pretty average."

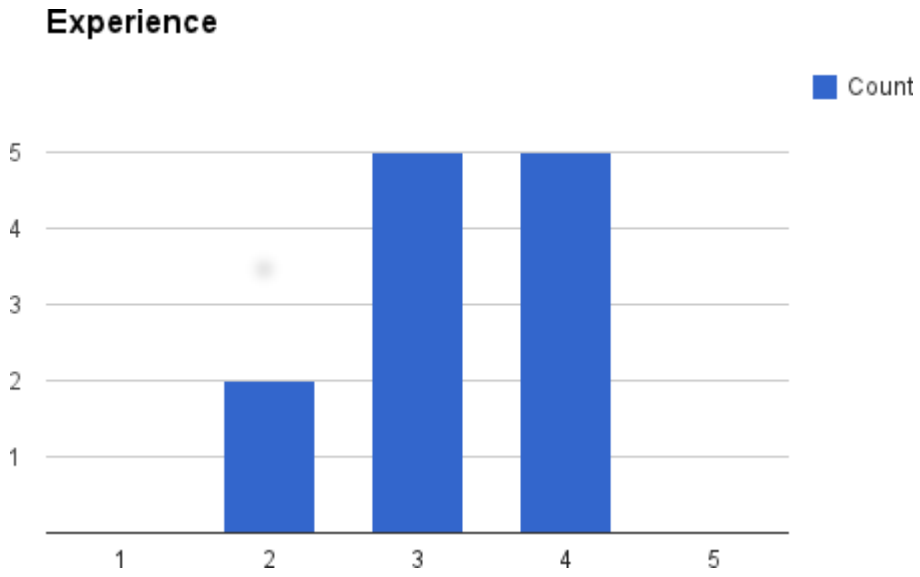


Age groups of test participants.

[4] Because I work on a university campus, I invited students, faculty, and staff to participate in a usability study. Volunteers were selected without preference for gender, age group, or level of experience. This reflects GNOME's preference to "target a broad range of users" (Day, email, 2014). Each tester was provided with a $5 gift card to the campus coffee shop in gratitude for their participation.



Gender representation of test participants.



Self-identified computer expertise of test participants.

Before each usability test session, every participant received a brief context of the usability study, explaining that this was a usability test of the *software*, not of the *user*. This introduction also encouraged testers to communicate their thought process; if searching for a print action, for example, the participant should state "I am looking for a 'Print' button." Testers were provided a laptop running a "live" image of GNOME 3.12 containing a set of example files, including the

text file used in the gedit scenario tasks. However, this version of GNOME proved unstable for most programs in the usability test. To mitigate the stability issues, the laptop was rebooted into GNOME 3.10 running on the Fedora 20 operating system to complete the scenario tasks for Web, Nautilus, Software, and Notes. In testing GNOME 3.10, each participant used a separate guest account that had been pre-loaded with the same example files as under GNOME 3.12. These example files also included items unrelated to the usability test, allowing participants to navigate these contents to locate files and folders required for the scenario tasks.

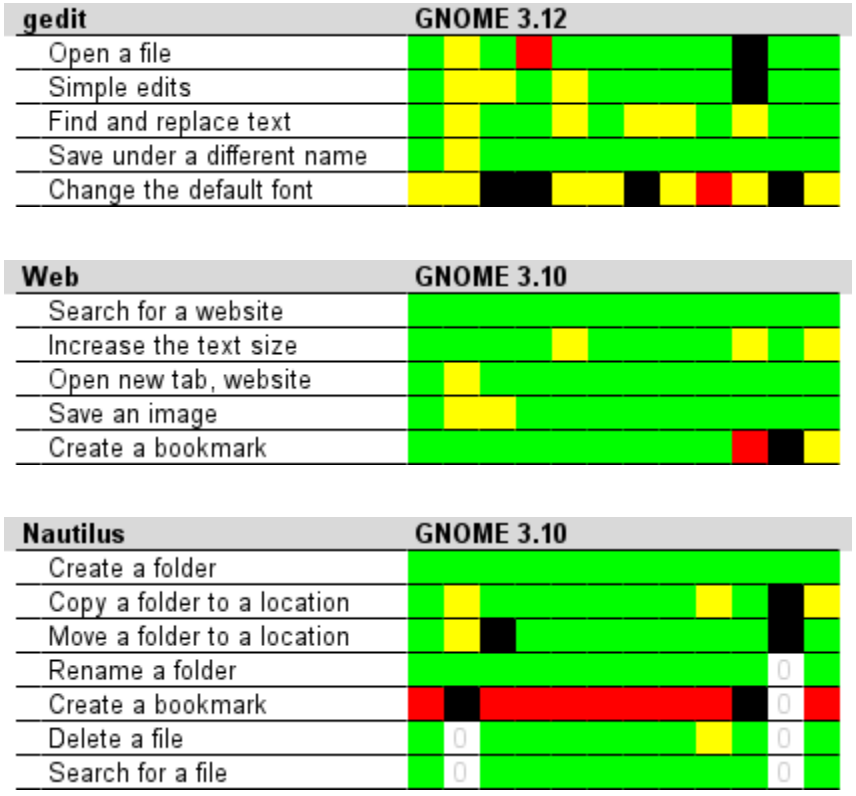
During the usability test, participants were presented with scenario tasks that represented typical user behaviors. With the exception of installing a program in Software, tasks did not use terminology from the program in describing the activity. For example, in the Web program, scenario tasks directed testers to “make the text bigger,” not to “increase the font size.”

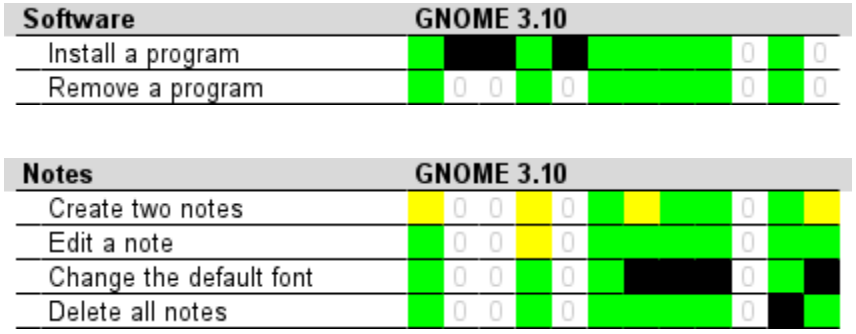
> 2. You don't have your glasses with you, so it’s hard to read the text on the BBC News website. Please make the text bigger on the website.

Subsequent to each usability test, each participant underwent a brief interview intended to probe their response to the tasks and programs. Overall, the usability test included 23 scenario tasks, which most testers completed in 50-55 minutes.

V. Usability test findings

A heat map neatly summarizes the results of the usability test. In the heat map, each scenario task is represented in a separate row, and each block in the row represents a tester's experience with that task. Green blocks indicate tasks that testers completed with little or no difficulty, and yellow blocks signify tasks that presented moderate difficulty. Red boxes denote tasks where testers experienced extreme difficulty or where testers completed tasks incorrectly. Black blocks indicate tasks the tester was unable to complete, while white boxes indicate tasks omitted from the test, usually for lack of time.





Heat map displaying usability test results of GNOME 3.12 and 3.10.

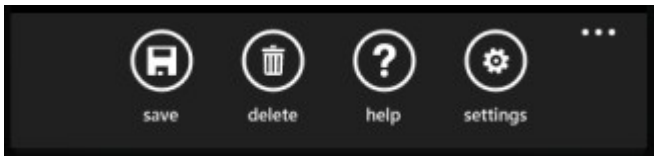
Assets

Participants generally experienced little or no difficulty with most usability test tasks. For example, all testers quickly and easily created and renamed folders in the file manager program, Nautilus. Similarly, testers experienced little difficulty when copying and moving folders, deleting files, and searching for files. In the gedit text editor, test participants encountered some difficulty in opening files, making edits, and saving files—but overall, participants completed these tasks. In Notes, testers also easily edited and deleted notes, although some participants found it somewhat challenging to create a second note in that program.

Challenges

Interestingly, all participants experienced significant issues with changing the default font in gedit (GNOME 3.12). A significant number of testers were unable to accomplish a similar task in Notes. In observing the tests, the testers typically looked for a “font” or “text” action under the gear menu. Many participants referred to the gear menu as the “options” or “settings” menu because of a previous affiliation with the gear icon and “settings” in other Mac OS X and Windows applications. These participants expected that

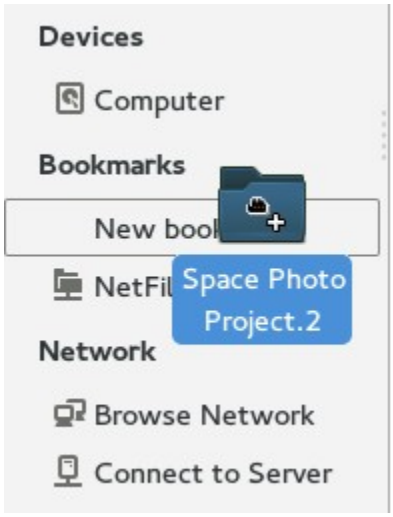
changing the font was an option in the program, and therefore searched for a “font” action under the gear or “options” menu. Part of this confusion stemmed from thinking of the text editor as though it were a word processor, such as Microsoft Word, which uses items in menus or on a toolbar to set the document font. This behavior was often exhibited by first highlighting all the text in the document before searching for a “font” action.



Example of the “Settings” gear menu (right) in Windows 8. [microsoft.com]

Example of the gear menu (right) in GNOME Nautilus file manager program.

In the Nautilus file manager, testers also experienced serious difficulty in creating a bookmark to a folder. In GNOME, this task is usually achieved by clicking *into* the target folder then selecting “Bookmark this Location” from the gear menu, or by clicking and dragging the intended folder onto “Bookmarks” in the left pane (see image). However, testers initially addressed this task by attempting to drag the folder onto the GNOME desktop. When interviewed about this response, almost all participants indicated that they prefer to keep frequently-accessed folders on the desktop for easier access. Most testers eventually moved the target folder into the Desktop folder in their Home directory, and believed they successfully completed the task even though the target folder did not appear on the desktop.



Nautilus: Dragging a folder onto "Bookmarks" in the left pane creates bookmark to that folder.

Testers also experienced difficulty when attempting "find and replace text" in gedit. In this task, testers employed the "Find" feature in gedit to search for text in the document. Experimenting with "Find," testers said they expected to replace at the same time they searched for text. After several failed attempts, testers were usually able to successfully invoke the "Find and Replace" action under the gear menu.

While the overall GNOME desktop was not part of the usability test, many testers experienced difficulty with the GNOME "Activities" hot corner. In the GNOME desktop environment, the "Activities" menu reveals a view of currently-running programs, and a selection of available programs. Users can trigger the "Activities" menu either by clicking the "Activities" word button in the upper-left corner of the screen or by moving the mouse into that corner (the "hot corner"). Although testers generally recovered quickly from the unexpected "hot corner" action, this feature caused significant issues during the usability test.

VI. Usability themes in open source software

This usability test uncovered multiple usability issues in GNOME 3.10 and 3.12. Several of these usability issues arise by virtue of how users apply meaning to icons, or from user preference in how to interact with menus or the desktop metaphor. Five features or *themes* emerged throughout the test, describing transcendental values of usability in open source software:

1. Familiar design

An important challenge in this usability test was Familiarity. GNOME is dissimilar from Mac or Windows, so everyone had to learn how to use GNOME. The differences are significant for users. Due to their prevalence, users have been "trained" to use mainstream desktop environments like Mac and Windows. Because the GNOME desktop behaves differently from these systems, testers were frustrated when attempting certain tasks such as changing the default font in gedit or Notes. The most serious difference was how GNOME manages the desktop itself. When asked to make a frequently-accessed folder more available by creating a "shortcut," many testers attempted to place the folder on the desktop. Mac and Windows support placing items on the desktop, but GNOME does not.

However, not everything in GNOME was a new experience for testers. Most notably, the Web program seemed familiar to many testers. Most of the tasks for Web caused little or no difficulty. Testers often attributed this ease of use to Web's similarity to other web browsers, such as Chrome or Firefox, and the testers responded to Web using lessons learned from these mainstream web browsers. For example, when asked to increase the

text size on a web page, a significant number of testers used the same Control-Plus keystroke shortcut used in Mac and Windows web browsers.

2. Consistent experience

Consistency was an important theme. Testers frequently reported that because all programs looked very similar, they were able to apply user interface experience from one program to other programs. Several testers reflected that subsequent tasks were easier than earlier tasks because they had become more familiar with the GNOME user interface.

3. Menu clarity

While testers were initially confused about having menus split across two different areas —the gear menu and the application menu—most eventually inferred the distinction. Several participants observed the gear menu (often referred to as “Options”) contained “program actions” while the application menu consisted of “top level” actions.

Interestingly, once testers discovered the application menu for certain program actions, many subsequently considered everything in the black GNOME “top bar” to be a menu. These participants often attempted to find program functions under the “Activities” button menu and the “system status area” menu, despite these being reserved for GNOME desktop purposes. (GNOME Shell Design).

GNOME top bar. The system status area is on the right, displaying wireless network connection, volume, and battery. The “Activities” button and gedit’s application menu are on the left.

4. Obvious results

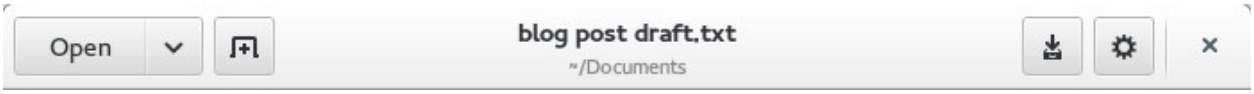
Throughout the usability test, some testers assumed they had not completed an action because GNOME did not give a very obvious indication that their action had an effect. A specific example is “Find and Replace” in gedit; when users successfully used this feature to replace instances of a word or phrase, the testers did not detect the acknowledgement of their action, and therefore believed the replace function did not find any such occurrences. Parallel to this example, GNOME also did not provide clear guidance when testers attempted to move folders onto the desktop; the default configuration of GNOME does not support this action, yet GNOME did not indicate this fact.

5. Visual relationships

In the usability test, testers relied on placement of user interface elements and icons to help them complete tasks. Testers were confused when buttons and menus were visually distant from the actions they affected. For example, several testers were unable to use the Software program to install a game because the “Install” button was in the header bar, disconnected from the program’s description in the middle of the screen. While the “Install” button was rendered in a different color, these testers did not locate the button, and instead clicked on another button that brought them to the program’s website.

Icons were another challenge for testers. Icons represent concepts and ideas, but meaning is fluid and variable according to the viewer (McCloud, 1993, p. 28). When icons are more abstract, they may be more difficult for users to understand. Therefore, designers must balance appearance with perception, or risk confusing the user. For example, testers

experienced difficulty saving a file in gedit and creating multiple reminders in Notes. In both of these instances, testers did not recognize the icon to effect the action (see images) on the program's header bar. In gedit, the "Save" icon is embodied by an arrow pointing down into a shape that suggests a cardboard box. In Notes, the "Back" button uses a "<" icon, a simple representation that carried little meaning for the testers.



Header bar for the gedit program. The "Save" icon is next to the gear menu.

Header bar for the Notes program. The "<" icon is the "Back" button.

Familiar design, consistent experience, menu clarity, obvious results, and visual relationships are good lessons in open source software usability. By following these five themes derived from the GNOME usability test, open source software developers can avoid problems in their own user interfaces, and improve ease of use; learn from these usability lessons rather than rediscover them. A developer's "creativity is better used in solving problems than in creating beautiful new impediments to understanding" (Spencer, 1998). This usability test suggests that open source software developers should leverage existing user interface paradigms, and remain consistent with other programs on the same platform, whether they are open source software or proprietary programs. Further, programs should use clearly labeled menus and employ recognizable icons. Finally, the program design must ensure that every action generates an obvious response, especially if that result indicates a failure.

VII. Conclusions

Open source software developers create an array of innovative programs, yet open source software is often plagued with poor usability. End users of many popular open source software programs share comments that blend praise of its functionality with criticism of its usability. Unfortunately, usability is often ignored in open source software development. Although some open source software projects have a maintainer who dictates a particular design aesthetic, most open source software projects focus on the functionality and the features, and give little thought to the user interface. This inattention to usability hinders users and damages the reputation of developers and open source software. Usability is overlooked or ignored at the peril of open source software.

If open source software is not attractive or easy to use, people will not want to use it. To resolve this dilemma, open source software developers must adopt usability testing to improve their product, enabling their audience to accomplish tasks quickly and easily. Usability must become part of the open source software process, not tacked onto a project at the end of its development lifecycle. In order to attract and retain users, to remain competitive and relevant, open source software developers must take the next step: bring usability testing into open source software development methodology.

The open source software usability challenge is cultural. To date, usability has been antithetical to open source software philosophy, where “Every good work of software starts by scratching a developer’s personal itch” (Raymond, 2000) and new features are incorporated based on need. Crafting new functionality takes priority, and open source developers rarely consider how end users will access those features.

However, a key strength of open source software development is the developer-user relationship; in open source software projects, the user community plays a strong role in testing new releases. Unfortunately, testers cannot rely on the typical user-testing cycle to provide sufficient user interface feedback. Usability researchers Nichols and Twidale comment that in open source software “Usability bugs may not be afforded the same status as functionality bugs,” (Nichols and Twidale) leading to an increasing divide between developer and user. A structural approach towards identifying usability issues in open source software addresses this dichotomy. Preston (2004) describes a variety of usability methods that open source software developers can easily adopt, making usability an integral part of the development process. These methods range from the complex to the simple, from a formal usability test to a basic observation. At the least, developers can learn a great deal simply by observing a few users interacting with the design (Redish, interview, 2012)..

Usability tests need not be performed in a usability laboratory environment. Open source software projects can leverage “crowdsourcing” with the open source community. For example, the Drupal web content management system streamed testers’ desktops as they participated in a usability test (Drupal, 2011), allowing developers worldwide to observe the usability tests remotely. The data gathered provided Drupal developers with a better understanding of user interface issues, and how to address them. Another simple method is a variation on the “flash mob,” a term suggested by Chisnell (2012). Similar to Preston’s Walk-Around Review, “flash mob” researchers intercept random people in a public space and ask them to respond to a few paper prototypes. If each participant provides a few minutes’ testing, a “mob” of such testers will generate valuable feedback in a short time.

Techniques like these mean that open source software developers do not need to be experts in usability testing to apply usability methodology to open source software projects. Developers need only observe users interacting with the program for usability issues to become clear. A handful of usability testers operating against a prototype provides sufficient feedback to make informed usability improvements. And with good usability, everyone wins.

Works Cited

Chisnell, Dana. (May 18, 2012). “Wilder than testing in the wild: usability testing by flash mob.” Web. Retrieved from http://usabilitytestinghowto.blogspot.com/2012/05/wilder-than-testing-in-wild-usability.html

Day, Allan. (January 17, 2014). “Re: GNOME user testing.” Email to author.

Drupal. (May 2011). “Drupal 7 UMN Usability Study.” Retrieved from https://archive.org/details/Drupal7UmnUsabilityStudy

Drupal. (June 1, 2011). “Report from the University of Minnesota Drupal Usability Testing.” Web. Retrieved from http://drupal.org/node/1175694

Dumas, Joseph S. and Redish, Janice C. (1993. Reprinted 1994). *A Practical Guide to Usability Testing: Revised Edition*. Portland, OR: Intellect.

GNOME. “Applications.” (Last updated December 17, 2013). Wiki. Retrieved from https://wiki.gnome.org/Apps/

GNOME. (Last updated February 12, 2014). “GNOME Human Interface Guidelines.” Wiki. Retrieved from https://wiki.gnome.org/Design/HIG/

GNOME. (Last updated November 25, 2013). “GNOME Shell Design.” Wiki. Retrieved from https://wiki.gnome.org/Projects/GnomeShell/Design

GNOME. Website. Retrieved from http://www.gnome.org/

Hall, Jim. (October 29, 2012). “Candidates for good usability.” Blog. Retrieved from http://opensource-usability.blogspot.com/2012/10/candidates-for-good-usability.html

Hall, Jim. (December 2013). “It’s about the User: Applying Usability in Open Source Software.” *Linux Journal*. 236.

James, Geoffrey. (1987). *The Tao of Programming*. Santa Monica, CA: InfoBooks.

McCloud, Scott. (1993). *Understanding Comics: The Invisible Art*. New York, NY: HarperCollins.

Nichols, David M.; Twidale, Michael B. “Usability Processes in Open Source Projects.” Graduate School of Library and Information Science, University of Illinois at Urbana-Champaign, IL, USA.

Open Source Initiative. (Version 1.9). “The Open Source Definition (Annotated).” Web. Retrieved from http://opensource.org/docs/definition.php

Preston, Alice. (January 2004). “Types of Usability Methods.” *Society for Technical Communication*. Vol. 10, No. 3. Retrieved from http://www.stcsig.org/usability/newsletter/0401-methods.html

Raymond, Eric S. (October 10, 2012). Personal interview.

Raymond, Eric S. (Version 3.0. Last modified September 11, 2000). “The Cathedral and the Bazaar.” Web. Retrieved from http://www.catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/

Redish, Janice. (December 10, 2012). “Re: Open source software and usability” Message to the author.

Redish, Janice. (October 15, 2012). Personal interview.

Spencer, Henry. (Modified July 14, 1998). “The Ten Commandments for C Programmers (Annotated Edition).” Web. Retrieved from http://www.lysator.liu.se/c/ten-commandments.html

Stallman, Richard M. (October 12, 2012). “Re: A question about Free software usability?” Message to the author.

Stallman, Richard M. (Version 1.111; February 20, 2012). “The Free Software Definition.” Web. Retrieved from http://www.gnu.org/philosophy/free-sw.html

Stallman, Richard M. “Why Open Source misses the point of Free Software.” Web. Retrieved from http://www.gnu.org/philosophy/open-source-misses-the-point.html

Walker, Tom. (March 10, 2010). “20 Most Popular Open Source Software Ever.” *Tripwire Magazine*. Web. Retrieved from http://www.tripwiremagazine.com/2010/03/20-most-popular-open-source-software-ever-2.html

Other references

Andersen, R. (2013). “The value of a reciprocal relationship between research and practice.” *CIDM Information Management News*.

Bach, Paula M. (2009). “Supporting the user experience in free/libre/open source software development.” PhD thesis, Pennsylvania State University. Retrieved from https://etda.libraries.psu.edu/paper/9880/5184

Benson, Calum; Müller-Prove, Matthias; Mzourek, Jiri. (2004). “Professional Usability in Open Source Projects: GNOME, OpenOffice.org, NetBeans.” *CHI EA ‘04* CHI ‘04 extended abstracts on Human factors in computing systems. pp 1083-1084. ACM New York.

Borchardt, Jan-Christoph. “Usability in Free Software.” Web. Retrieved from http://jancborchardt.net/usability-in-free-software

Diamandis, Peter. (February 2012). “Abundance is our future.” TED2012. Retrieved from http://www.ted.com/talks/peter_diamandis_abundance_is_our_future.html

Fennell, Chad. (October 29, 2012). Personal interview.

Friedman, Thomas. (2007). *The World Is Flat 3.0: A Brief History of the Twenty-first Century*. New York, NY: Picador / Farrar, Straus, and Giroux.

Hahn, Harley and Stout, Rick. (1994). *The Internet Complete Reference*. Berkeley, CA: Osborne McGraw-Hill.

Hall, Jim. (November 9, 2009). “Cultivating Open Source Software.” Blog. Retrieved from http://www.freedos.org/jhall/blog/?id=20091109-155503

Nielsen, Jakob. “How Many Test Users in a Usability Study?” Web. Retrieved from http://www.useit.com/alertbox/number-of-test-users.html

Perens, Bruce. (1999). “The Open Source Definition.” In DiBona, Chris and Ockman, Sam (Eds.) *Open Sources: Voices from the Open Source Revolution*. O’Reilly Media. Chapter also available from http://oreilly.com/catalog/opensources/book/perens.html

Rosencrans, Nick. (November 2, 2012). Personal interview.

Rosen, David. (October 30, 2012). Personal interview.

Rude, C.D. (2009). “Mapping the research questions in technical communication.” *JBTC*, 23(2).

Simoes, Fabiana. (2013) “How to not report your UX bug.” Conference presentation. GUADEC 2013. Retrieved from http://www.superlectures.com/guadec2013/how-to-not-report-your-ux-bug

The GNOME Usability Project. GNOME. Web. Retrieved from https://live.gnome.org/UsabilityProject/

Torvalds, Linus and Diamond, David. (2001). *Just for Fun: The Story of an Accidental Revolutionary*. New York, NY: HarperBusiness.

U.S. Department of Health & Human Services. “Usability Basics.” Web. Retrieved from http://www.usability.gov/basics/

Appendix: Participant survey

Prior to the usability test, each participant was asked to respond to this brief survey, intended to capture basic demographic information.

1. Your age: (please circle one)

- 18 - 24
- 25 - 34
- 35 - 44
- 45 - 54
- 55 - 64
- 65 - 74

2. Your gender: (fill in the blank)

_____

3. Please indicate (circle) your level of computer expertise:

1. I know very little about computers
2. I know some things, but not a lot
3. I am pretty average
4. I am better than most
5. I am a computer expert

Institutional Review Board determination: "As this is a usability test, and you are not collecting private identifiable data about the people themselves, this does not qualify as Human Subjects Research and does not need any further review by the UMN IRB."

Appendix: Scenario tasks

Each test participant was presented with a set of tasks intended to demonstrate typical behavior of the GNOME desktop applications, and thereby the GNOME design patterns. This list was reviewed with the GNOME Project design team prior to usability testing.

Gedit

1. You want to finish writing a draft of a blog post that you are using in a project. You start the Gedit text editor (this has been done for you).

Please open the file **blog post draft.txt** from the **Documents** folder, into the Gedit text editor..

2. You realize that you got a few details wrong. Please make these edits:
In the first paragraph, change the dash ("—") to a semicolon (";")

from this: **Relationships are currency—you**

to this: **Relationships are currency; you**

In the second paragraph, change "me" to "others" - relationship to me. to relationship to others.

About 2/3 into the document, there's a list of the "4 I's" of relationships, but the first two items are out of order. Put these into the correct order, so the list reads like this:

1. **Initiate**
2. **Inquire**
3. **Invest**
4. **Inspire**

When you are done, please save the file. You can use the same filename.

3. Some of the names are incorrect in the file. Please replace every occurrence of **Applejack** with **Fluttershy**, and all instances of **Rainbow Dash** with **Twilight Sparkle**.

When you are done, please save the file. You can use the same filename.

4. You'd like to make a copy of the note, using a different name that you can find more easily later. Please save a copy of this note as **Leadership lessons.txt** in the **Documents** folder.

For the purposes of this exercise, you do not need to delete the original file.

5. You decide the text in the editor is difficult to read, and you would prefer to use a different style. Please change the text style to be **Nimbus Mono L**, 12 point.

Web

1. You would like to see what's happening in the news. Your favorite news website is **BBC News**, but you don't remember the website's address.

Find the **BBC News** website, and bring up the site.

2. You don't have your glasses with you, so it's hard to read the text on the BBC News website. Please make the text bigger on the website.

3. You now would like to see what's new with your favorite "open source" project, FreeDOS. The website's address is **http://www.freedos.org/**

However, you want to keep an eye on the news while you are looking at the FreeDOS website. In a new site, navigate to the FreeDOS website.

4. You decide to download a copy of the FreeDOS program screenshot, the sample program image on the right side, under "Welcome to FreeDOS." This image might come in handy for a document you are working on (writing the document is not part of this exercise).

Save a copy of this image as **freedos.png** in the **Pictures** folder.

5. You would like to visit the FreeDOS website later, but don't want to keep typing in the address every time. Create a shortcut for the FreeDOS website.

When you create the shortcut, name it simply **FreeDOS**.

Nautilus

1. You are about to start working on a new project, and you would like to keep all your files in the same folder.

Please create a new folder called **My Project** in the **Documents** folder.

2. Your research partner has given you a bunch of files that you can use in your project, but you'll need to copy them from his USB flash drive to your computer. Copy the **Project001** folder (all the folders and files) from the USB flash drive (provided) into the **My Project** folder in the **Documents** folder.

3. Oops! You realize that the files in your new project will be photos, so you prefer to keep them under the **Pictures** folder instead of **Documents**.

Please move the **My Project** folder from **Documents** to **Pictures**.

4. Your project now has a new name, so you decide to rename the project folder to use the new name.

Please rename the **My Project** folder to **Space Photo Project**.

5. As you work on your project, you expect to go back to the folder frequently. A shortcut to the folder would be handy, rather than having to navigate to it each time.

Create a shortcut to **Space Photo Project**.

6. You don't need the **email to mom.txt** file that was accidentally included in the project files. Delete the **email to mom.txt** file in the **Mars** photos folder. This is located under **Space01**, then under **Photos**, then under **Mars photos**.

7. You would like to update a draft of your Space article, which you started several weeks ago. You don't remember where you last saved the file, so you need to find it. Find the file **Draft of space article.txt**. When you find it, open it.

Software

1. Your friend has told you about an interesting arcade-style game that you'd like to try out on your computer. Install the game **Robots**.

~~2. After installing **Robots**, try it out to see what it's like.~~ *(This task was omitted after the first test when the tester became distracted from the usability test when playing the game.)*

3. This is the last task for Software. Please uninstall the game **Robots**.

Notes

1. You need to type up a few quick reminders for yourself, so you don't forget to do them later. Enter these two reminders into the Notes program. Match the formatting as best as you can:

First reminder:

Don't forget: Jeff's surprise party this Thursday. Check with Mark.

Second reminder:

Things to buy at the grocery store:

- **Milk**
- **Eggs**
- **Cheese**

2. You decide to be more clear in your reminder about Jeff's surprise party, so you don't forget the time. Update that note to say this:

Don't forget:

Jeff's surprise party this Thursday.
Get together at 5:30.
Party at 7:00.

Check with Mark.

3. You decide the text in Notes is difficult to read, and you would prefer to use a different style. Please change the default text style to be **PT Sans**, 14 point.

4. This is the last task for Notes. Please delete all notes you may have created during this part of the usability test.

Appendix: Testing GNOME design patterns

Day recommended several GNOME design patterns to include in usability testing (email, 2014):

> I'd be particularly interested in looking at selection mode, header bars, notifications, content overviews (browsing within the content apps), full screen, app menus and gear menus. The patterns look a bit different in some places, so it might be nice to compare, say, selection mode in Software compared with selection mode in Clocks.

Day suggested a list of ten updated GNOME applications that utilize the new design patterns concepts:

1. Clocks
2. Contacts
3. Documents
4. gedit *(development version only)*
5. Maps
6. Music
7. Notes
8. Photos
9. Software
10. Web

This updated usability test began by understanding the target users. GNOME designer Allan Day shared that GNOME does not identify a particular class of users, such as developers or computer experts. Rather, GNOME targets all users (email, 2014):

> I think you are right … in that we target a broad range of users. We tend to think about it like this:
>
> 1. Core experience - this is the operating system, it includes the shell and the rest of the "bare" system. It is a very "thin" platform for running apps. ie. We provide as little functionality outside of applications as possible.

> 2. Core apps - these cover the basic things that almost everyone needs, as well as some essential utilities that are needed by the system. You get a browser, apps for viewing files and other content, handy utilities like a calculator, and so on.
>
> 3. 3rd party apps - these are how diverse users fulfil their more specialist requirements. Applications are how you extend the functionality of your system.
>
> I wouldn't necessarily say that the core apps are targeting a particular kind of user. Rather, they are targeting what the majority of users have in common.

As a result, the design vision for GNOME addresses an inclusive audience. Users may be anyone. GNOME seeks the common denominator. Accordingly, a usability test that examines GNOME should not objectify a particular kind of test participant. Testers should represent a combination of expertise, genders, and age groups.

In order to compare results between this usability test and my previous usability test conducted with Firefox and GNOME 3.4, the new test design included GNOME's Nautilus, Web, and gedit. Nautilus is the GNOME file manager, Web is a web browser similar to Firefox or Google's Chrome, and gedit is a text editor comparable to Notepad on Windows. An analysis of the design patterns used in several GNOME programs resulted in a list of programs that share several patterns with Nautilus, Web, and gedit. For example, while most GNOME programs utilize header bars, application menus, content overview, and selection mode, not all use the gear menu or notifications. However, the gear menu is heuristically interesting, as it provides a "second menu" by which users may perceive access to other program functions, separate from the application menu. Notes utilizes several of these design patterns. Additionally, Software employs both the view switcher and selection mode patterns. The five programs included in this iteration of the usability test were:

1. gedit
2. Notes
3. Nautilus
4. Software
5. Web

Four of these programs were suggested by Day. The other program, Nautilus, allows a direct comparison with previous usability test results. According to Day, the GNOME Project is hesitant to make further changes to Nautilus at this time (email, 2014).

Appendix: Literature Summary

The present study is broader in scope and approaches the issue from an academic perspective. The connections between usability in open source software and academic study are compelling. As suggested by Andersen (2013), academia and practice may establish a reciprocal relationship to enhance both areas. In this cycle, academia provides accessible, actionable research that is published in media frequented by practice. Such research allows practice to advance, elevating the state of the art. As the cycle continues, both academia and practice benefit.

While the field of "Usability Testing" has received significant attention, insufficient research has been applied to usability in open source software. Usability studies of open source software projects are somewhat rare in academic literature. A few examples become prominent in their frequent citation. David M. Nichols and Michael B. Twidale have authored several articles on the subject, including "Usability Processes in Open Source Projects" (University of Illinois at Urbana-Champaign). Calum Benson, Matthias Müller-Prove, and Jiri Mzourek describe usability testing of open source software projects ("Professional Usability in Open Source Projects: GNOME, OpenOffice.org, Netbeans"). Other research in this field draws important conclusions, but appears less frequently in the literature. Paula M. Bach (College of Information Sciences and Technology, Pennsylvania State University) explored "Supporting the User Experience in Free/Libre/Open Source Software Development" in her Ph.D. thesis (2009). Similarly, Jan-Christoph Borchardt (Information Design, Stuttgart Media University) examined low-cost usability testing in independent free and open source software projects (2011) and is now an open document (CC BY-SA 3.0) summarizing "Usability in Free Software."

Other research concerning usability in open source software has typically been conducted informally or sponsored by corporate entities interested in adopting or marketing open source software. In a presentation at the GNOME Users' And Developers' European Conference (GUADEC), held annually in cities around Europe, Fabiana Simoes discussed issues surrounding the capture and reporting of UX (User eXperience) issues in open source software (2013).

While not intended as a complete reference, the following summarizes relevant literature related to usability in open source software:

Academic resources

About open source software

Andreasen, M. S.; Nielsen, H. V.; Schrøder, S. O.; Stage, J. (2006). "Usability in Open Source Software Development: Opinions and Practice." *ISSN 1392 – 124X Information Technology and Control*, 2006, Vol. 35, No. 3A. Retrieved from http://itc.ktu.lt/itc353/Stage353.pdf

Bach, Paula M. (2009). "Supporting the user experience in free/libre/open source software development." PhD thesis, Pennsylvania State University. Retrieved from https://etda.libraries.psu.edu/paper/9880/5184

Benson, Calum; Müller-Prove, Matthias; Mzourek, Jiri. (2004). "Professional Usability in Open Source Projects: GNOME, OpenOffice.org, NetBeans." *CHI EA '04* CHI '04 extended abstracts on Human factors in computing systems. pp 1083-1084. ACM New York. Retrieved from http://mprove.de/script/04/chi

Frishberg, N., Dirks, A. M., Benson, C., Nickell, S., Smith, S. (2002). "Getting to Know You: Open Source Development Meets Usability." In: CHI 2002, Minneapolis, MI. ACM Press. Retrieved from http://doi.acm.org/10.1145/506443.506666 and http://www.iol.ie/~calum/chi2002/

Nichols, David M.; Twidale, Michael B. (2003). "The Usability of Open Source Software." *First Monday* 8(1). Retrieved from http://firstmonday.org/article/view/1018/939

Nichols, David M.; Twidale, Michael B. “Usability Processes in Open Source Projects.” Graduate School of Library and Information Science, University of Illinois at Urbana-Champaign, IL, USA.

Nichols, David M.; Twidale, Michael B. (2005). “Exploring Usability Discussions in Open Source Development.” *Proceedings of 38th Annual Hawaii International Conference on System Sciences*, HICSS-38, Track 7: “Internet and the Digital Economy,” p.198c.

Trudelle, P. (2002). “Shall We Dance? Ten Lessons Learned from Netscape’s Flirtation with Open Source UI Development.” Web copy of CHI2002 article. Retrieved from http://iol.ie/~calum/chi2002/peter_trudelle.txt

About usability in general

Ayobami, Akanmu Semiu;Hector, Okere P.; Hammed, Adebambo. (2 November 2012). “Current Issues of Usability Characteristics and Usability Testing (2012).” *Proceedings of International Conference on Behavioural & Social Science Research* (ICBSSR), Kampar, Malaysia. Retrieved from http://ssrn.com/abstract=2170600

Raskin, J. (1994). “Intuitive Equals Familiar.” *Communications of the ACM*. 37:9, September 1994, pg. 17. Retrieved from http://asktog.com/papers/raskinintuit.html

Non-academic resources

About open source software

Balazs, B. (2011). “Designer vs. Developer?” Web. http://opensource-usability-labs.com/tine20/2011/03/18/designer-vs-developer OFFLINE.

Benson, Callum. (July 20, 2001). “GNOME usability test report now available.” Email to mailing list. Retrieved from http://lwn.net/2001/0726/a/gnome-usability-report.php3

Borchardt, Jan-Christoph. “Usability in Free Software.” Web. Retrieved from http://jancborchardt.net/usability-in-free-software

Çetin, G.; Erkan, K.; Verzulli, D.; Ozirkovskyy, L. (2006). “Usability Involvement in F/OSS Projects.” Contracted professional research study. Retrieved from http://tossad.org/content/download/1381/6882/file/tOssad_D19_V1.pdf

Day, A. (July 8, 2010). “User Experience Advocates.” Blog. Retrieved from http://afaikblog.wordpress.com/2010/07/08/user-experience-advocates/

Faaborg, A. (2010b). “Firefox 3.5 & 3.6 Menu Item Usage Study.” Web. http://testpilot.mozillalabs.com/testcases/menuitemusage OFFLINE.

Pennington, H. (April, 2002). “Free software UI.” Web. Retrieved from http://www543.pair.com/rhp/free-software-ui.html

Reitmayr, E.; Balazs, B.; Mühlig, J. (2006). “Integrating Usability with Open Source Software Development: Case Studies from the Initiative OpenUsability.” In özel, B., Çilingir, C. B., Erkan, K. (Eds.). *Oss 2006 tOssad workshop proceedings* (p. 65–72). http://tossad.org/tossad/events/tossad_2006/proceedings

Simoes, Fabiana. (2013). “How to not report your UX bug.” Video of conference presentation (topic based on Master’s thesis). GUADEC 2013. Retrieved from http://www.superlectures.com/guadec2013/how-to-not-report-your-ux-bug

Smith, S., Engen, D., Mankoski, A., Frishberg, N., Pedersen, N., and Benson, C. (2001). “GNOME Usability Study Report.” Sun Microsystems, Inc. http://developer.gnome.org/projects/gup/usertesting.html OFFLINE.

Thomas, M. P. (August 1, 2008). “Why Free Software has poor usability, and how to improve it.” Web. Retrieved from http://web.archive.org/web/20090625141639/http://mpt.net.nz/archive/2008/08/01/free-software-usability

Wilms, T. (March 10, 2011). Design-Boost. Blog. Retrieved from http://thorwil.wordpress.com/2011/03/10/design-boost/

Zer-Aviv, Mushon. (September 1, 2010). “The Case For Open-Source Design: Can Design By Committee Work?” Retrieved from http://www.smashingmagazine.com/2010/09/01/the-case-for-open-source-design-can-design-by-committee-work/

About usability in general

Barnum, Carol M. (2010). *Usability Testing Essentials: Ready, Set … Test!* Elsevier.

Chisnell, Dana. (October 6, 2009). “Usability Testing Demystified.” Web. Retrieved from http://alistapart.com/article/usability-testing-demystified

Dumas, Joseph S. and Redish, Janice C. (1993. Reprinted 1994.) *A Practical Guide to Usability Testing: Revised Edition*. Portland, OR: Intellect.

Faaborg, A. (2010). "Quantifying Usability: How injecting usability principles into standard bug tracking software can reshape how your organization approaches UX design." Web. http://uxmag.com/strategy/quantifying-usability OFFLINE.

Hawley, Michael. (July 24, 2012). "Modifying Your Usability Testing Methods to Get Early-Stage Design Feedback." Web. Retrieved from http://www.uxmatters.com/mt/archives/2012/07/modifying-your-usability-testing-methods-to-get-early-stage-design-feedback.php

Johansson, R. (May 2, 2005). "Usability testing without a budget." Web. Retrieved from http://www.456bereastreet.com/archive/200505/usability_testing_without_a_budget/

Krug, S. (2008). "The least you can do about usability." Video of conference presentation. Retrieved from http://blip.tv/business-of-software/steve-krug-on-the-least-you-can-do-about-usability-1566021

Lewis, J. R. (August 24, 2006). "Usability Testing." Technical report. IBM Software Group. Retrieved from http://drjim.0catch.com/usabilitytesting-ral.pdf

Mathis, L. (2011). "Designed for Use." Extract from book. Retrieved from http://media.pragprog.com/titles/lmuse/prototype.pdf

Morrow, J. L. (July 6, 2011). "User Testing in the Wild: Joe's First Computer Encounter." Blog. Retrieved from http://jboriss.wordpress.com/2011/07/06/user-testing-in-the-wild-joes-first-computer-encounter/

Nelson, T. (1990). "The Right Way to Think About Software Design." In Laurel, B. (Ed.), *The Art of Human-Computer Interface Design*. Reading: Addison-Wesley. Extract from book. Retrieved from http://ui.korea.ac.kr/Board/Upload/theorder_n2.pdf

Nielsen, J. (January 1, 1995). "Ten Usability Heuristics for User Interface Design." Web. Retrieved from http://www.nngroup.com/articles/ten-usability-heuristics/

Nielsen, J. (January 1, 1995). "Severity Ratings for Usability Problems." Web. Retrieved from http://www.nngroup.com/articles/how-to-rate-the-severity-of-usability-problems/

Nielsen, J. (August 5, 2001). First Rule of Usability? Don't Listen to Users. Web. Retrieved from http://www.nngroup.com/articles/first-rule-of-usability-dont-listen-to-users/

Nielsen, J. (March 19, 2000). Why You Only Need to Test with 5 Users. Web. Retrieved from http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

Nielsen, J. (1994). "Heuristic evaluation." In Nielsen, J. and Mack, R.L. (Eds.), *Usability Inspection Methods*. New York, NY: John Wiley & Sons.

Siegel, D. (2010). "Announcing the User Experience Advocates Project." Web. http://davidsiegel.org/announcing-ux-advocates OFFLINE.

Siegel, D. (2009). "Paper Cut." Web. http://davidsiegel.org/papercut OFFLINE.

Websites of interest

"Usability.gov." Web. Retrieved from http://www.usability.gov/

"Open Usability." Web. http://www.openusability.org/ OFFLINE.

GNOME Usability Project. "Usability Tests." Wiki showing test results from 2001, 2005, 2009, 2010. Retrieved from https://wiki.gnome.org/UsabilityProject/UsabilityTests

"Usability." Blog. Retrieved from https://groups.drupal.org/usability

Other references

Raymond, Eric S. (Last updated: Revision 1.57, September 11, 2000). "The Cathedral and the Bazaar." Web. Retrieved from http://www.catb.org/~esr/writings/homesteading/

Stallman, R. (1986; Last updated January 12, 2014). "The Free Software Definition." Web. Retrieved from http://www.gnu.org/philosophy/free-sw.html

Acknowledgements

My wife, Sara Rouner.

My advisor, Dr. Ann Hill Duin.

My faculty reviewer, Dr. MaryElizabeth Bezanson.

Peer reviewers Wonjong Choi, Christopher Cocchiarella, Changsoo Lee, and Heather Sigstad.

GNOME designers Allan Day, William Jon McCann, and Jakub Steiner.

Invited commenters Dr. Lee-Ann Breuch and Dr. Janice Redish.