A call for bioimaging software usability

Anne E Carpenter, Lee Kamentsky & Kevin W Eliceiri

Bioimaging software developed in a research setting often is not widely used by the scientific community. We suggest that, to maximize both the public's and researchers' investments, usability should be a more highly valued goal. We describe specific characteristics of usability toward which bioimaging software projects should aim.

© 2012 Nature America, Inc. All rights reserved.

Imaging in biology has recently seen a revolution in scope and scale, necessitating developments in image informatics on several fronts. New, sophisticated bioimaging techniques yield large, heterogeneous, multidimensional data sets that need to be viewed, analyzed, annotated, queried and shared. Conversion to digital-based imaging modalities raised the bar in terms of the quantification that biologists need and expect for their images. The exploding volume of images collected in modern experiments necessitates automated analysis. Microscopy techniques of increasing complexity and functionality have become widespread and often require new modes of analysis. Image-derived data are even beginning to be used as a research and clinical biomarker, where subtle changes are detectable by computational means but cannot be confirmed by eye.

These developments have presented unique challenges, leading to the rapid and continuing creation of algorithms and software packages for bioimaging[1]. Unfortunately, serious usability problems with academic bioimaging software limit its impact on the community. Often, publicly funded software is only useful within the research group that created it, leading to redundant effort and wasted research funding. By contrast, bioimaging software is 'usable' when both biologists and developers can accomplish their research goals without inordinate effort or expert knowledge. Usable software often results from very close collaboration between users and programmers, such that the tools do not suffer from lack of software engineering expertise or lack of real-world applicability.

Anne E. Carpenter and Lee Kamentsky are at the Imaging Platform, Broad Institute of Massachusetts Institute of Technology and Harvard, Cambridge, Massachusetts, USA. Kevin W. Eliceiri is at the Laboratory for Optical and Computational Instrumentation, University of Wisconsin at Madison, Madison, Wisconsin, USA.
e-mail: anne@broadinstitute.org or eliceiri@wisc.edu

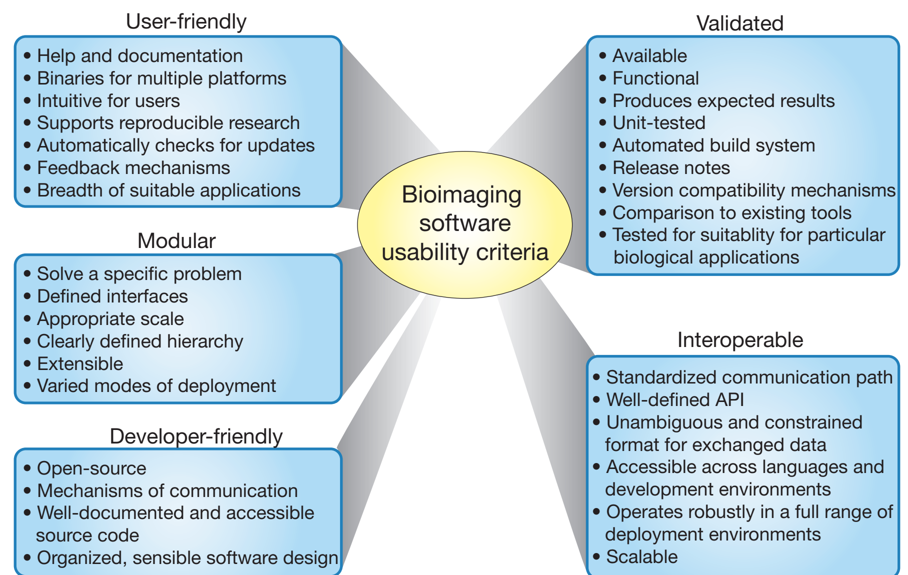

Figure 1 | Software for bioimaging should aim to meet specific usability criteria.

Here we advocate that researchers intending to create software that is usable, high-impact and helpful to a broad bioimaging community aim for excellence according to several key criteria (**Fig. 1**). These criteria are good practice for any software and have been the subject of much recent research and discussion[2,3]; we focus on bioimaging in this Commentary owing to the pressing needs of this community.

We do not claim that every software project ought to meet all of these usability criteria, nor that our own projects do. Time and funding are not available in most academic groups to meet these software engineering 'best practices', and those writing code for bioimaging software are rarely software engineers. As well, we recognize that exploratory computational projects and development of new algorithms would suffer if held to a high standard of usability from the outset, before their use or application is even determined. Still, although there is absolutely a place for exploratory research, especially for training purposes in computer science, it alone is insufficient. Producing

an algorithm, a methods paper and/or code is of limited utility to most biologists, so researchers claiming that the output of their computational work will serve the broader community must meet a higher standard.

Thus, we advocate that every software project claiming broad impact should consider these criteria and that reviewers and funding agencies should reward researchers who strive toward them. We suggest that funding agencies request that these criteria be addressed (though not necessarily met) in a 'plans for software sharing and usability' section of proposals' resource-sharing plans. We recognize that implementing and maintaining high usability in software can be challenging, that resources are limited, and that choices must be made among many priorities for software projects. However, we strongly advocate that researchers, institutions and funding agencies place a higher value on usability in bioimaging software.

User-friendly

Although modern training in biology increasingly involves exposure to software, advanced computational skills are certainly not universal. Further, bioimaging involves a breadth of computational knowledge that is difficult for a single researcher to master. Therefore, except for toolkits or libraries solely for other developers' use, it is critical that bioimaging software be usable by non-experts.

At a minimum, the software should be packaged with some form of help and documentation, such as installation and use instructions and written or video tutorials. Software projects should consider having mailing lists or online forums for support and to gather user feedback. The software should be available as binaries for multiple platforms. Requiring researchers to build the software from source code is time-consuming even for computer experts and is impossible for many in the target audience. The software design itself should be intuitive for users. Aside from common sense, substantial research in 'usability engineering' is available to guide software developers to create software that can be quickly learned and efficiently used[4,5].

The software should support reproducible research, with mechanisms allowing an analysis to be reproduced with minimal effort and thus shared with colleagues or published with a paper (http://www.reproducibleresearch.net/)[6–9]. A key component of reproducibility is provenance. The software output should indicate the configuration, settings and version of the software that created the results, and prior versions should be available for download and confirmation. Users also benefit when software automatically checks for updates and has feedback mechanisms to inform developers of problems and user needs. To prevent users' wasted effort, the breadth of suitable applications of the software should be described—that is, the extent to which the software is expected to work outside the narrowly constrained application for which it was developed.

Developer-friendly

Many features of user-friendly software also make the software developer-friendly, reducing the amount of time developers spend assisting users with routine issues such as installation and basic use. Developers also have distinct needs that should be considered for software projects intended to serve as the basis for others' work. Although exceptions exist, it is usually desirable but also practical for publicly funded projects to be provided with an open-source licensing option (http://www.opensource.org/), given that most projects will not be moneymakers and an open-source license is not necessarily incompatible with commercial use and licensing. Having the latest version of the code base available is valuable in a scientific setting, allowing other researchers to understand its function and to improve the code[10]. There should be transparent mechanisms of communication among developers, such as forums for discussions of future plans for development. Well-documented source code and organized, sensible software design can ease other developers' work and preserve the value of the project if the original developers cease working on the project.

Interoperable

Although a single software package that solves all bioimaging problems may sound appealing, the community benefits from individual research groups working in different areas, from acquisition to storage to analysis and visualization[1]. To serve the biological community, however, these individual software packages must effectively interact with each other, allowing researchers to create workflows using complementary components that best serve their needs. The bioimaging software community has recently begun to come together to work on interoperability among some of the more popular packages. Recent examples include a US National Institutes of Health (NIH) American Recovery and Reinvestment Act–funded effort to link ImageJ[11] with CellProfiler[12], the Bio-Formats project[13] and ImgLib[14].

Interoperability offers many benefits. It reduces tedious and error-prone effort, avoids manual manipulation of data between processing steps by individual researchers and avoids the creation of custom bridges with limited applicability to others. Interoperability yields tremendous flexibility, such that biologists can choose the individual software packages, imaging instrumentation and computing hardware best suited to their project. Aside from reducing redundant effort, it encourages creative approaches and rapid adaptation to new application areas by allowing individual research groups to focus on developing software for a piece of the bioimaging pipeline while relying on interoperability with established software for other portions of the pipeline. Creating useful bridges exposes the users of each software package to the other software package, increasing the user base of each. This yields higher-quality, more robust software because features are tested more frequently, and developers from both projects can add features and fix bugs.

But interoperability also presents challenges to developers. It often requires major time commitment and effort as well as coordination between groups focused on their own core area. Securing resources to establish connections between software to increase scientists' efficiency rather than adding major new functionality is a major challenge. Although the use of multiple programming languages complicates connections between software, programming tools for cross-language communication are becoming more common.

Interoperability can be implemented in several degrees. At one end of the spectrum, one software package can be made to export images or data in a format that another software package can read as input. For example, CellProfiler can export image-derived data in a format readable by several other software packages for downstream data analysis. Creating a more seamless interface, where the user has direct access within one software package to the functionality of the second, requires much more effort. An example is CellProfiler's ability to run an ImageJ macro[12] or an Ilastik

© 2012 Nature America, Inc. All rights reserved. npg

machine-learning algorithm[15] from within a CellProfiler pipeline.

The key to interoperability is for software to have a standardized communication path and a well-defined application-programming interface (API). Ideally, the software should have a communication path accessible across languages and development environments, and the mechanism used to connect should be self-documenting and discoverable. Traditionally, C header files and APIs have been used to accomplish this, but increasingly, web interfaces and protocols are used to connect applications. Individual applications can also implement their own API autodiscovery mechanisms. For instance, ImageJ 2.0 plug-in developers can now publish the parameters of their plug-ins, and those plug-ins can then be integrated with any application that understands the protocol.

Interoperability is also facilitated by the software using an unambiguous and constrained format for exchanged data so that the receiver can validate their implementation against test inputs. An example of this is the Open Microscopy Environment (OME) Data Model[16], which uses an Extensible Markup Language (XML) schema to represent microscopy data. Interoperable software operates robustly in a full range of deployment environments and scenarios. The code should be thread-safe and ideally stateless so that it can be run in parallel, and the software should be scalable for large experiment sizes, with mechanisms to leverage a computer cluster.

Modular

A major theme of modern software development is software modularity. Modules generally contain an algorithm that solves a specific problem, often targeted to a particular domain, and are written to conform to defined interfaces. As opposed to monolithic software, modular software provides users and developers several benefits. Modules of appropriate scale—large enough to have a coherent purpose but small enough to mix and match in new ways—offer flexible adaptation to new applications as opposed to a software package built to solve a particular, narrowly defined problem.

There is a balance to be struck here: a risk of modularity is that the library of options might become overwhelming to the user, especially if divided into units whose individual functions are too fine-grained for a non-expert to understand their purposes. This can be mitigated by providing example pipelines or macros, in which small-scale modules are configured and combined properly to accomplish a particular goal. A key aspect of modularity is to have a clearly defined hierarchy, where the foundation is the algorithm, and successive layers can combine algorithms to support users in a narrower domain or with less technical expertise.

Because a single component of modular software can be improved or adapted to new applications by others, modular software is by nature extensible, allowing it to serve as the basis of future work in the field. Modular software reduces redundant developer effort by allowing new functionality to be added without creating entirely new systems. This is not just a time-saver; the scientific community benefits from having multiple options available in an already stable, sustainable infrastructure, for rapid testing and accurate comparison. Although often researchers prefer to create a new piece of software from scratch for ownership and visibility purposes, adding functionality to an existing tool increases the dissemination, and thus validation of the work, and builds on biologists' existing knowledge.

Modular software enables widespread and varied modes of deployment because the same modules can be accessed in many ways (web, client, server, cluster, graphical user interface or command line) and even from different software packages, supporting interoperability. Modularity also supports effective software design because it encourages organized code, unit tests and categorizing functionalities.

The benefits of modularity are illustrated by the OME Consortium's Bio-Formats, a modular library for loading and saving image data and metadata (http://www.loci.wisc.edu/software/bio-formats/)[13]. Bio-Formats has a standardized and well-defined programming interface that developers can easily access from their own software. Not only does the widespread use of Bio-Formats by software developers (in over 20 applications) and biologists (more than 20,000 laboratories) result in great resource savings, the Bio-Formats library itself is a better-performing library because of this widespread modular use, which has yielded improvements in performance and file-format support.

Validated

By validated, we mean that software should be tested in several ways that are relevant to users. Most importantly, software must be made available. Even if requesters must register their identity or agree to specific license terms before downloading, hosting the software online enables faster, more reliable access compared to when software is 'available upon request', which unfortunately some institutions require. Providing reliable long-term availability is now straightforward through online repositories such as Github, SourceForge and Google Code. This prevents ignored requests and defunct web addresses[17,18]. The software should be maintained in functional form for current operating systems. If this is not possible, availability of the source code becomes even more important, as future developers can use it to build binaries. It is fairly common that published software cannot be installed and started, perhaps because of changes in operating systems or hardware, or because necessary components of the software are missing. Debugging such issues is often beyond the capabilities of the typical biologist end-user. Hosting an instance of a software application in the cloud or as a virtual machine avoids many installation problems and allows users to run the software remotely.

Beyond these bare minimum criteria to reliably obtain, start up and use software, validation also encompasses testing for accuracy. The software should produce expected results and should be provided with an example test analysis and necessary images or data that can be loaded to verify that the software is functioning correctly for the user. Such examples can also serve as a helpful starting template for the biologist to adapt for their analysis.

At a more fundamental level, software should be unit-tested, where each unit of code or algorithm in a program has a corresponding test suite that compares the code's actual output against expected values using inputs that cover all conditions and branches[19]. Unit tests should not be considered a luxury for large projects only; they usually save developers' and users' time in the long run by aiding in debugging. Tests are especially helpful in combination with an automated build system, where the software is compiled periodically (for example, nightly or upon every change to the source code). Every time an automated build occurs, the unit tests can be run and the resulting output can be compared to what is expected to ensure that units of the software continue to yield reproducible, correct results. Because

© 2012 Nature America, Inc. All rights reserved. npg

new versions of software often have altered functionality or produce different results because of improvements in the underlying algorithms, software should be provided with release notes to document changes and version compatibility mechanisms to inform the user of which version of the software was used to create particular results and also to load old analyses and data into new versions of the software.

Validation encompasses the suitability of software for particular goals. Most researchers consider a paper describing an algorithm or software methodology to be incomplete unless there is a conscientious comparison to existing tools in terms of features and performance. 'Conscientious' is of course subjective. As an example, an ideal comparison of image-analysis algorithms should compare a proposed new algorithm using a publicly available test image set upon which others have already tested their algorithm to the best of their ability. If an appropriate image set is not available, algorithm developers should at least provide the image set and ground truth so others can test their algorithm on the same images[20]. Repositories are emerging for this purpose—for example, the Broad Bioimage Benchmark Collection (http://www.broadinstitute.org/bbbc)[21]. The literature is rife with algorithms that are presented with purely subjective, qualitative comparison to alternate algorithms or with comparison to existing algorithms that have not been applied properly, sometimes without even complete description of the configuration of the alternate algorithm.

Lastly, users benefit when software is tested for suitability for particular biological applications. It can save substantial time if the software documentation indicates whether it is suited to handle particular image file formats, imaging modalities, cell types and so on. In particular, it can be helpful for users to know whether software has been tested only on a few image sets from the creator's laboratory or on a variety of images representing heterogeneous use cases and environments.

Conclusions

Creating software that meets the usability criteria we outlined here requires substantial time and effort; contributing a module to an existing infrastructure is usually more efficient. Still, there are cases where more substantial software development is needed, and this is difficult to fund through standard research grants focused on answering particular biological questions. Tenure committees and other assessors at academic institutions also often do not value usable software as worthwhile research output. Reviewers with a biological background are typically interested in research that tests a particular biological hypothesis; many of the usability-specific criteria above are difficult to justify in such a context. Conversely, reviewers from a computer science background tend to focus on algorithmic novelty rather than practical utility and again, find it difficult to justify an emphasis on usability. Regrettably, this yields hundreds of publicly funded proof-of-principle papers describing algorithms that do not make their way out of the literature and into the biology laboratory.

Supporting the new development and ongoing maintenance of software thus requires that funding agencies and grant proposal reviewers support substantial funding dedicated to software development *per se*. Fortunately, funding agencies have increasingly recognized the value of usable software to the biomedical research enterprise. Direct support for open-source software development has increased in recent years, helping to counteract the trend for reviewers and funding agencies to prize novelty of features or innovation over more practical concerns of usability.

Both the US National Science Foundation (NSF) and the NIH have specific grant programs that support the creation and maintenance of software, in addition to their long-standing commitments to biology-oriented computational research in general. The NIH created a program in 2002 entitled "Continued Development and Maintenance of Software" (PAR-11-028, http://grants.nih.gov/grants/guide/pa-files/PAR-11-028.html). This exemplary program has funded ~200 software projects, including some widely used in bioimaging (**Table 1**). The NSF, for its part, recently announced the Software Infrastructure for Sustained Innovation (SI^2) program, which supports developing modular software with an emphasis on good software engineering practices and usability principles. Although the SI^2 program supports all fields of science and is too new to evaluate, it indicates NSF's commitment to usability and may indicate future increased expectations for all NSF-funded software projects. In addition to these programs, we advocate for complementary new small grant programs simply for the basic maintenance of popular bioinformatics tools.

Calculating the return on investment for funding federal software is challenging, but consider the NIH Image (now called ImageJ) project[11]. The project was created and maintained over 15 years by essentially one software developer supported by the NIH, yet its impact has been tremendous, having nucleated thousands of volunteer programmers to develop plug-ins and yielding 7,000 visitors to its website every day and more than 210,000 citations as of 7 June 2012 in Google Scholar. Much of the use is by NIH-funded researchers whose research is made more efficient and accurate through use of the software, saving the NIH many-fold the cost of its development.

In addition to supporting projects to improve software's usability, funding agencies should refrain from funding researchers with poorly thought-out plans or a poor track record in this area. We recommend that funding agencies require that any proposal involving algorithm or code

Table 1 | Bioimaging software: NIH's Continued Development and Maintenance of Software program

Software	Bioimaging area	First year of program funding
ScanImage	Laser scanning acquisition	2003
IMOD	Electron tomography analysis	2005
BioImage Suite	Medical imaging analysis	2006
EMAN	Electron microscopy analysis	2006
BRAINS	Brain image analysis	2007
Cell Centered Database	Image database	2007
Neuroimaging in Python	Neuroimaging analysis	2007
MicroManager	Light microscopy acquisition	2008
HAMMER	Image warping and registration	2009
CellProfiler	High-throughput image analysis	2010
SPIDER	Electron microscopy analysis	2010
ITK-SNAP	Image segmentation	2011

© 2012 Nature America, Inc. All rights reserved. npg

development includes in its 'Resource Sharing Plan' a plan for software sharing and usability. This section would describe the project's plans related to usability criteria, such as those we outlined in **Figure 1**. As in the 'Resource Sharing Plan' section that is currently required for all NIH proposals, and the 'Data Management and Sharing' section required for NSF proposals, requiring a plan for software sharing and usability does not mean that all criteria must be met and certainly not that they be met in a particular way. It simply clarifies what the researchers are proposing and holds them accountable to their commitments. This will assist reviewers in assessing the likely impact of the proposal and provides a venue for the research team to describe their track record in producing usable software.

It might initially be thought that publicly funded open-source projects would infringe upon commercial entities' interests or threaten the health of the commercial software marketplace, but this has typically not been the case. On the contrary, interactions between open-source projects and commercial entities in bioimaging have largely been mutually beneficial. In fact, most open-source bioimaging software is not developed to create an inexpensive or free alternative to similar commercially available software; instead, the goal is to develop solutions for emerging needs that are beyond the current scope of market demand. Many companies benefit from the freely available advancements that stem from academia's ability to rapidly respond to changing fields, especially new instrumentation and techniques, in a way that is closely connected to the biological problems at hand. Owing to the complexity of methods, the wide array of data types and the need for rapid development and improvement, these imaging methods often require an open sharing of analysis approaches well suited to the nonprofit research environment, at least in their early stages. Many of the commercially available software tools that are now often used for image acquisition, analysis and visualization had their developmental start in the academic laboratory. Many commercial software products interface with open-source software; improvements to the usability of publicly funded software projects benefit both. Other types of relationships exist as well. For example, the OME project creates open-source software and a companion company, Glencoe Software, works closely with companies wishing to use the code in their own programs, often contributing code improvements back to the open-source project.

Widespread support for usability therefore requires cultural shifts: researchers, reviewers, institutions and funding agencies must appreciate that devoting resources to usability has a major impact on the scientific community rather than valuing solely hypothesis-driven research or algorithm novelty. The scientific payoff in doing so could be tremendous: devoting more attention and resources to usability promises to yield enormous benefit to biological research worldwide.

ACKNOWLEDGMENTS

We thank members of our research groups and our software projects for helpful feedback on the manuscript, in particular CellProfiler cofounder T.R. Jones. This work was supported in part by US National Institutes of Health grants R01 GM089652 (to A.E.C.) and RC2 GM092519 (to K.W.E.), but the opinions expressed are solely those of the authors.

COMPETING FINANCIAL INTERESTS

The authors declare no competing financial interests.

1. Eliceiri, K.W. *et al. Nat. Methods* **9**, 697–710 (2012)
2. Baxter, S.M., Day, S.W., Fetrow, J.S. & Reisinger, S.J. *PLoS Comput. Biol.* **2**, e87 (2006).
3. Fogel, K. *Producing Open Source Software* (O'Reilly Media, Inc., 2005).
4. Bolchini, D., Finkelstein, A., Perrone, V. & Nagl, S. *Bioinformatics* **25**, 406–412 (2009).
5. Rimmer, J. *J. Audiov. Media Med.* **27**, 6–10 (2004).
6. Vandewalle, P., Kovacevic, J. & Vetterli, M. *Signal Processing Magazine, IEEE* **26**, 37–47 (2009).
7. Mesirov, J.P. *Science* **327**, 415–406 (2010).
8. Peng, R.D. *Science* **334**, 1226–1227 (2011).
9. Ioannidis, J.P.A. & Khoury, M.J. *Science* **334**, 1230–1232 (2011).
10. Ince, D.C., Hatton, L. & Graham-Cumming, J. *Nature* **482**, 485–488 (2012).
11. Schneider, C.A., Rasband, W.S. & Eliceiri, K.W. *Nat. Methods* **9**, 671–675 (2012).
12. Kamentsky, L. *et al. Bioinformatics* **27**, 1179–1180 (2011).
13. Linkert, M. *et al. J. Cell Biol.* **189**, 777–782 (2010).
14. Preibisch, S., Tomancak, P. & Saalfeld, S. *ImageJ User and Developer Conference* **1**, 72 (2010).
15. Sommer, C. *et al. Proc. IEEE Int. Symp. Biomed. Imaging* 230 (2011).
16. Goldberg, I.G. *et al. Genome Biol.* **6**, R47 (2005).
17. Schultheiss, S.J., Münch, M.C., Andreeva, G.D. & Rätsch, G. *PLoS ONE* **6**, e24914 (2011).
18. Veretnik, S., Fink, J.L. & Bourne, P.E. *PLoS Comput. Biol.* **4**, e1000136 (2008).
19. Zhu, H., Hall, P.A.V. & May, J.H.R. *ACM Comput. Surv.* **29**, 366–427 (1997).
20. Ruusuvuori, P. *et al. BMC Bioinformatics* **11**, 248 (2010).
21. Ljosa, V., Sokolnicki, K. & Carpenter, A.E. *Nat. Methods* **9**, 637–638 (2012).

© 2012 Nature America, Inc. All rights reserved. npg