Title: A comprehensive analysis of the usability and archival stability of omics computational tools and resources

Serghei Mangul[1,2$#], Thiago Mosqueiro[2$], Dat Duong[1], Keith Mitchell[1], Varuni Sarwal[3], Brian Hill[1], Jaqueline Brito[4], Russell Jared Littman[1], Benjamin Statz[1], Angela Ka-Mei Lam[1], Gargi Dayama[7], Laura Grieneisen[7], Lana S. Martin[2], Jonathan Flint[5], Eleazar Eskin[1,6], Ran Blekhman[7,8]

[1] Department of Computer Science, University of California Los Angeles, 580 Portola Plaza, Los Angeles, CA 90095, USA

[2] Institute for Quantitative and Computational Biosciences, University of California Los Angeles, 611 Charles E. Young Drive East, Los Angeles, CA 90095, USA

[3] Indian Institute of Technology Delhi, Hauz Khas, New Delhi, Delhi 110016, India

[4] Institute of Mathematics and Computer Science, University of São Paulo, São Paulo, Brazil

[5] Center for Neurobehavioral Genetics, Semel Institute for Neuroscience and Human Behavior, University of California Los Angeles, 760 Westwood Plaza, Los Angeles, CA 90095, USA

[6] Department of Human Genetics, University of California Los Angeles, 695 Charles E. Young Drive South, Los Angeles, CA 90095, USA

[7] Department of Genetics, Cell Biology and Development, University of Minnesota, 321 Church St SE, Minneapolis, MN 55455, USA

[8] Department of Ecology, Evolution, and Behavior, University of Minnesota, 100 Ecology Building, 1987 Upper Buford Cir, Falcon Heights, MN 55108, USA

$ - These authors contributed equally to the paper

- Corresponding author

Abstract

Developing new software tools for analysis of large-scale, biological data is a key component of advancing computational, data-enabled research. Scientific reproduction of published findings requires running computational tools on data generated by such studies, yet little attention is presently allocated to the usability and archival stability of computer code encapsulated as computational software tools. Scientific journals require data and code sharing, but none currently require authors to guarantee software usability and long-term archival stability of newly published tools. We developed an accurate estimation of the accessibility of computational biology software tools by performing an empirical analysis of usability and archival stability of 24,490 omics software resources published from 2000 to 2017. We found that 26% of all omics software resources are currently not accessible through URLs published in the paper. Among the tools selected for our comprehensive and systematic usability test, 49% were deemed "difficult to install," and 28% of the tools failed to be installed due to problems in the implementation. Moreover, for papers introducing new software, we found that the number of citations significantly increased when authors provided an easy installation process for published software. We propose for incorporation into journal policy several practical solutions for increasing the widespread usability and archival stability of published bioinformatics software.

Introduction

During the past decade, the rapid advancement of genomics and sequencing technologies has generated an enormous amount and diversity of new algorithms in computational biology [1,2]. In the last 15 years, the amount of available genomic sequencing data has doubled every few months. In order to analyze this unprecedented volume of genomic data [5], many life-science and biomedical researchers are leveraging computational tools to solve complex biological problems and subsequently lay the essential groundwork for the development of novel clinical translations [6]. The exponential growth of genomic data has therefore reshaped the landscape of contemporary biology, making computational tools a key driver of scientific research [3,4].

Novel challenges and standards arise as computational and data-enabled research become increasingly popular in biology. One such challenge is computational reproducibility—the ability to replicate published findings by running the same computational tool on the data generated by the published study [7–9]. In order to scientifically reproduce published findings, a researcher must be able to run the computational tool with original settings and parameters on data generated by such studies. While several journals have introduced enforcements for the sharing of data and code, there currently are no effective requirements to promote usability and long-term archival stability of software tools. Limited software usability and archival stability of computational tools can ultimately impair our ability to reproduce published results.

The synergy between computational and wet-lab researchers and the reproduction of published results are especially productive when software developers distribute their tools as packages that are easy to use and install [10]. Ideally, computational tools for genomic analysis would not require of the user extensive knowledge in computer science for installation and usage. Given the emergence of new tools released each year, comparatively inadequate presentation and distribution of an otherwise advantageous software package could limit its scientific utility [11]. The computational biology community is now building a consensus on the importance of encouraging software development that is both computationally efficient and easy to use [10,12–14]. Primary principles behind successful computational biology software include quality and reusability of the source code, a factor that helps avoid reimplementing previously developed solutions to recurrent problems [11,12].

Widespread support for software usability promises to have a major impact on the scientific community [15], and practical solutions have been proposed to guide the development of scientific software [12,14,16,17,18,19,10,13]. Such solutions represent a crucial first response to the growing 'software crisis' of inefficiency and redundancy, a dilemma driven by the limited usability yet the prolific online availability of many computational biology tools published each year. While the scale of the 'software crisis' in computational biology has yet to be estimated, the bioinformatics community warns that poorly maintained or improperly implemented tools will ultimately hinder progress in big data-driven fields, such as genomics and systems biology [4,5,20].

Challenges to effective software development and distribution in academia

Successfully implementing and distributing software for scientific analysis involves numerous unique challenges that have been previously outlined by other scholars [10,12–14]. In particular, fundamental differences between software development workflows in academia and in industry challenge the usability and archival stability of novel tools developed by academics. Academic developers produce and test the majority of new computational biology tools, yet, in comparison to industry employees, the academic worker has access to fewer resources for producing usable, archivally stable packages.

First, software developers in industrial settings receive considerably more resources for developing user-friendly tools than their counterparts in academic settings [23]. The public or private software is developed by large teams of software engineers that include specialized user experience (UX) developers. In academic settings, software is developed by smaller groups of researchers who may lack formal training in software engineering, particularly UX and cross-platform design. Many computational tools lack a user-friendly interface to facilitate the installation or execution process [11]. Developing an easy-to-use installation interface is further complicated when the software relies on third-party tools that need to be installed in advance, called 'dependencies.' Installing dependencies is an especially complicated process for researchers with limited computational knowledge. Even a stable online presence cannot guarantee widespread usability of such software tools; life science and medical researchers cannot explore all potential options for analyzing genomic or other types of biological "big data" with a software that lacks an easy-to-use installation interface. Well-defined UX

standards for software development could help software developers in computational biology promote widespread implementation and use of their newly developed computational tools.

Second, companies efficiently distribute industry-produced software using dedicated company units or contractors—services that universities and scientific funding agencies do not typically provide for academic-developed software. The computational biology community has adopted by default a pragmatic, short-term framework for disseminating software development [24]. In academia, the dissemination model of new software consists of publishing a paper describing the software tool in a peer-reviewed journal. So-called "methods papers" are dedicated to explaining the rationale behind the novel computational tool and demonstrating with sample datasets the efficacy of the tool. Supplemental materials such as detailed instructions, tutorials, dependencies, and source code are made available on the internet and included in the published paper as a URL. The quality, format, and long-term availability of supplemental materials varies among software developers and is subject to less scrutiny in the peer-review process compared to the published paper itself. This approach limits the usability of software tools and ultimately hinders the community's ability to evaluate the tools in benchmarking studies [25].

Third, industry-developed software is supported by teams of software engineers dedicated to developing and implementing updates for as long as the software is considered valuable to the community. Many software developers in academia do not have access to mechanisms that could ensure continuous maintenance and long-term archival stability of published tools. Journals require publicly accessible URLs when publishing a computational tool, but there is presently no standard approach for ensuring long-term archiving of web content.

For example, many published tools in computational biology are hosted on academic web pages that become inactive with time, sometimes only months after initial publication. These software packages are typically developed by small groups of graduate students or postdoctoral scholars who, considering the temporary nature of such positions, cannot maintain such websites and software for longer periods of time.

Fourth, computational biology software developers in academia receive more incentive and support to develop new tools than to maintain existing tools. Once published, the structures of funding, hiring, and promotion in academia offer the developer little incentive for continuous, long-term development and maintenance of existing software tools and databases [21]. Software developers can lose funding for even the most widely-used tools. Loss of external funding may halt and even discontinue software development, potentially impacting the research productivity of studies that depend on these tools [22]. Halted software development also hinders the ability to reproduce results from published studies that use discontinued tools.

Archival stability of published computational tools and resources

The World Wide Web provides a platform of unprecedented scope for data and software accessibility, yet long-term preservation of online resources remains a largely unresolved problem [26]. Published software tools are made accessible through the Uniform Resource Locator (URL), which is typically provided in the abstract or main text of the paper and is often assumed to be a practically permanent locator. However, a URL may become inactive due to removal or reconfiguration of web content. This phenomenon is described by various terms, including 'death of URL' [27] or 'lost Internet Reference' [28]. At the onset, the World Wide

Web promised the virtually infinite availability of digital resources; in practice, many digital resources are lost.

Multiple studies have identified across various biomedical journals the deterioration of long-term archival stability of published software tools [20,27–31]. In order to begin assessing the current software crisis in computational biology, we comprehensively evaluated the archival stability of computational biology tools used in 51,236 biomedical papers published across 10 relevant peer-reviewed journals over a span of 17 years, from 2000 to 2017. Out of the 51,236 examined papers, 13.6% contained at least one URL in their abstracts, and the other 38.3% contained URLs in the body of the paper. To ensure that the identified URL corresponds to an active software tool or database, we inspected 10 neighboring words for specific keywords commonly used, including "pipeline", "code", "software", "available", "publicly", and others (See Methods Section). Complete details on our methodology for extracting the URLs, including all parameters and thresholds, are provided in the Supplementary Methods.

We used a web mining approach to test 26,631 published URLs that our survey identified. Of all identified URLs, 4.2% were unreachable because of connection timeouts, and 24.4% were 'broken' (i.e., 404 HTTP status). The threshold for allotted time may bias results; we manually verified URLs reported with the timeout error code (**Figure S1**).

Next, we grouped the URLs by the year in which the computational biology tool was referenced by the corresponding publication. As expected, the time since publication is a driving factor for URL archival stability (Kruskal-Wallis, p-value $<10^{-16}$). 31.9% of the software before 2012 are unavailable; whereas only 13.6% of the recent software (after 2012) are unavailable (**Figure 1a**). After 2014, we observe a drop in the absolute number of archivally

unstable resources (**Figure 1b**). Despite the strong decline in the percentage of archivally unstable resources with time, there are still 200 archivally unstable resources published every year. The data and scripts for reproducing the plots in **Figure 1** are available at https://github.com/smangul1/good.software/wiki/.

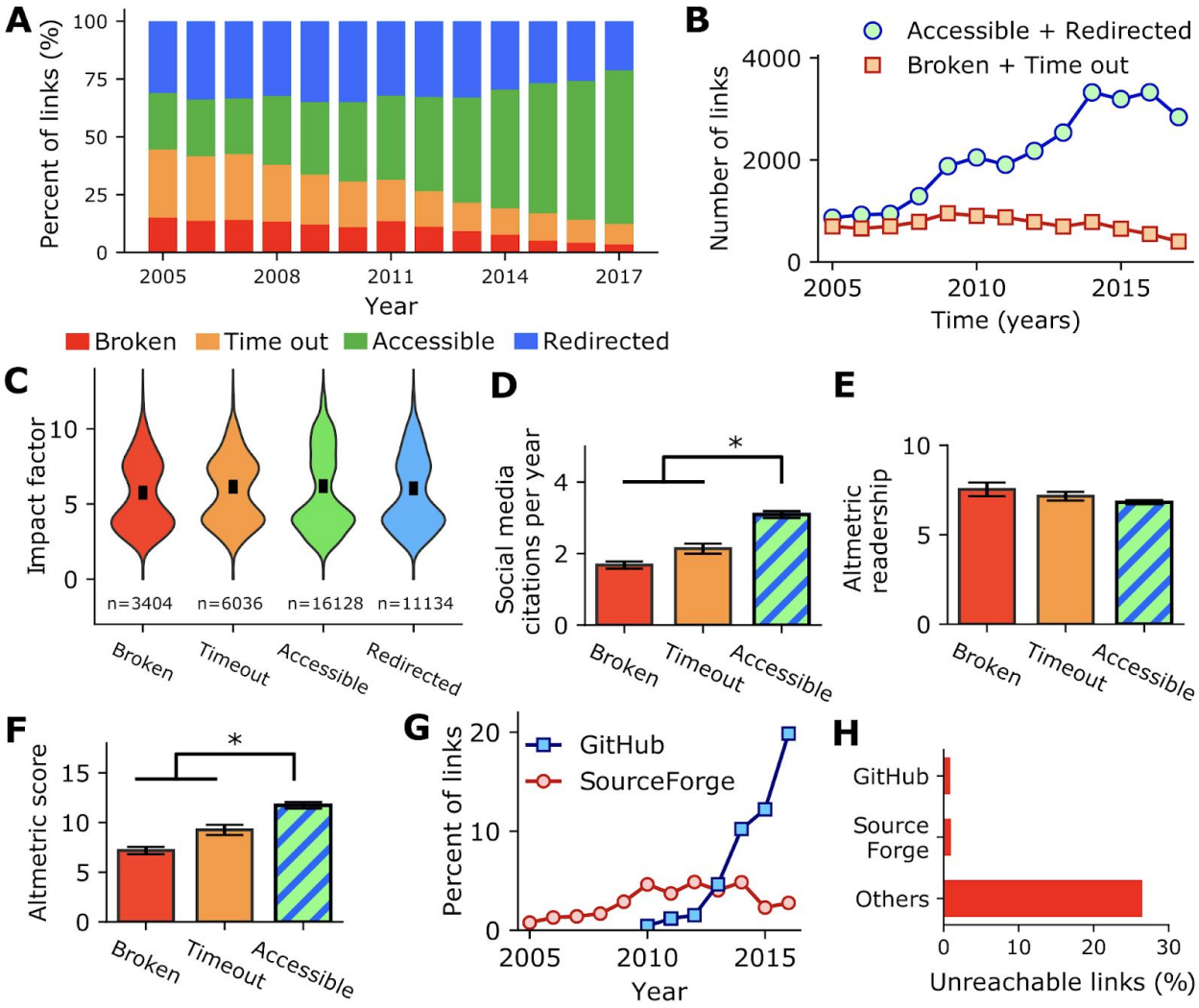

Figure 1. Archival stability of 24,490 published URLs across 10 systems and computational biology journals over the span of 17 years. **(a)** Accessibility status of 7,010 URLs parsed from the abstracts of published peer-reviewed papers. We categorized unreachable URLs into two groups: unreachable due to connection timeout ('Timeout', orange color), and unreachable due to error (i.e., 404 HTTP status) ('Broken', red color). We separately categorized accessible URLs that use redirection ('Redirected', blue color) and accessible URLs that do not redirection 'Accessible' (green color). Percentages of each category (y-axis) are reported over a 12-year span (x-axis). **(b)** Distribution of 7,010 abstract URLs (extracted from the abstract of the published papers) across 12 years. URLs are categorized as 'accessible + redirected' and 'broken + timeout'. Numbers of URLs per category reported across 12 years. **(c)** No effect of the journal impact factor on the availability of published links was observed. **(d)** Accessible links exhibit increased citations in social media (e.g., blog posts, twitter feeds, etc) per year compared to 'broken' and 'timeout' links (Kruskal-Wallis, p-value $= 10^{-256}$). **(e)** Altmetric readership score (y-axis) is not significantly different across URL categories (x-axis). **(f)** Accessible links exhibit increased Altmetric score compared to 'broken' and 'timeout' links (Kruskal-Wallis, p-value $= 10^{-15}$). **(g)** The proportion of unreachable links (due to connection timeout or due to error) stored on web services designed to host source code (e.g., GitHub and SourceForge) and 'Other' web services. **(h)** URLs hosted on web services that are designed to host source code exhibit decreased the fraction of unreachable links ('GitHub' vs 'Others' p-value=10^{-249}; SourceForge vs 'Others' p-value=$10^{-7,}$ Fisher-exact test).

A published URL can often be relocated to another URL, which is connected to the original URL via redirection. We found that 25% of active URLs are redirected to new URLs. However, 26% of updated URLs for published software are still not connected to the original published URLs (that is, there is no redirection to the current software URL). This disconnection is a consequence of the static nature of journal publications, which does not allow updating information in the published content.

Similarly to the results of previous studies [30], the results of our survey show no effect of the journal impact factor on the availability of published links (**Figure 1e**). Prior research demonstrates that the availability of published bioinformatics resources has a significant impact on citation counts [30]. In addition to those generally accepted measures of scientific impact, we have assessed the effect of software availability on complementary metrics of impact, such as measures of social media mentions, media coverage, and public attention (**Figure 1f-h**). We found that papers with accessible links exhibit increased engagement by readers in social media, reflected in a significantly higher number of citations in social media platforms (e.g., blog posts, twitter feeds, etc) per year and an increased Altmetrics score [32] when compared to papers with 'broken' and 'timeout' links (Kruskal-Wallis, H=492, *p*-value $= 10^{-256}$).

In addition, we tested the impact of using websites designed to host source code, such as GitHub and SourceForge, on the archival stability of bioinformatics software. These websites have been used by the bioinformatics community since 2001, and the proportion of software tools hosted on these sites has grown substantially, from 5% in 2012 to 20% in 2017 (**Figure 1g**). We find that URLs pointing to these websites have a high rate of accessibility; 99% of the links

to GitHub and 96% of the links to Sourceforge are accessible, while only 72% of links hosted elsewhere are accessible (**Figure 1h)**.

Our results suggest that the computational biology community would benefit from such approaches, which effectively guarantee permanent access to published scientific URLs. Specifically, several key principles emerge that promise to positively impact the availability of published bioinformatics resources, including the number of citations and social media references. In addition, bioinformatics tools and resources stored on web services designed to host source code have a significantly higher chance of remaining accessible.

Tool usability

We have developed a computational framework capable of systematically verifying the accessibility and usability of published software tools. We applied this framework to 99 randomly selected tools across various domains of computational biology (**Method Section**). We engaged undergraduate and graduate students to run the installation test using a standardized protocol (**Figure S2**); we recorded the time required to install the tools and other important features, allowing up to two hours per software package. In total, 72 hours of installation time was required to install 99 tools. We categorized a tool as 'easy to install' if it could be installed in 15 minutes or less; 'complex installation' if it required more than 15 minutes but was successfully installed before the two hour limit; and 'not installed' if the tool could not be successfully installed within two hours (**Table S1** and **Figure 2**).

We determined that 57.1% of the selected tools failed the 'automatic installation test', where the tester is required to strictly follow the instructions provided in the manual of the software tool (Methods; **Figure 2a**). In most tools (52%), the automatic installation test finished in fewer than 15 minutes (**Table S1**), as no additional commands outside of the manual were required. For the tools failing the test, we performed manual intervention where the tester was allowed to install missing dependencies and modify code to resolve the installation error. On average, it took an additional 70 minutes to run commands not provided in the installation instructions to successfully install the tool, which resulted in a significant increase of installation time (Kruskal-Wallis, p-value=4.7×10^{-9}; **Figure 2b**). Manual intervention was unsuccessful for 66% of the tools that initially failed the automatic installation test; failed manual installation was due to numerous issues, including hard coded parameters, invalid folder paths or header files, and usage of unavailable software dependencies.

Next, we assessed the effect of the ease of installation on the popularity of tools in the computational biology community by investigating the number of citations for the paper describing the software tools. We find that tools which we were able to install had significantly more citations compared to tools which we were not able to successfully install within two hours (**Figure 2c**; Kruskal-Wallis, p-value=0.032). These results suggest, perhaps not surprisingly, that tools which are easier to install are more likely to be adopted by the community.

In addition, we aimed to see whether the documentation within the code affects the installation time. Considering the proportion of commands that are undocumented (estimated as a ratio between the executed commands and commands in the manual), we find that tools with easier installation have a significantly lower percentage of undocumented commands

(**Figure 2d**; Kruskal-Wallis, p-value=2.3×10^{-6}). Considering a significant increase of installation time and a low rate of success for tools failing automatic installation test, we argue that reliance on manual intervention to successfully install and run computational biology tools is an unsustainable practice. Software developers would benefit from ensuring a simple installation process and providing adequate installation instructions.

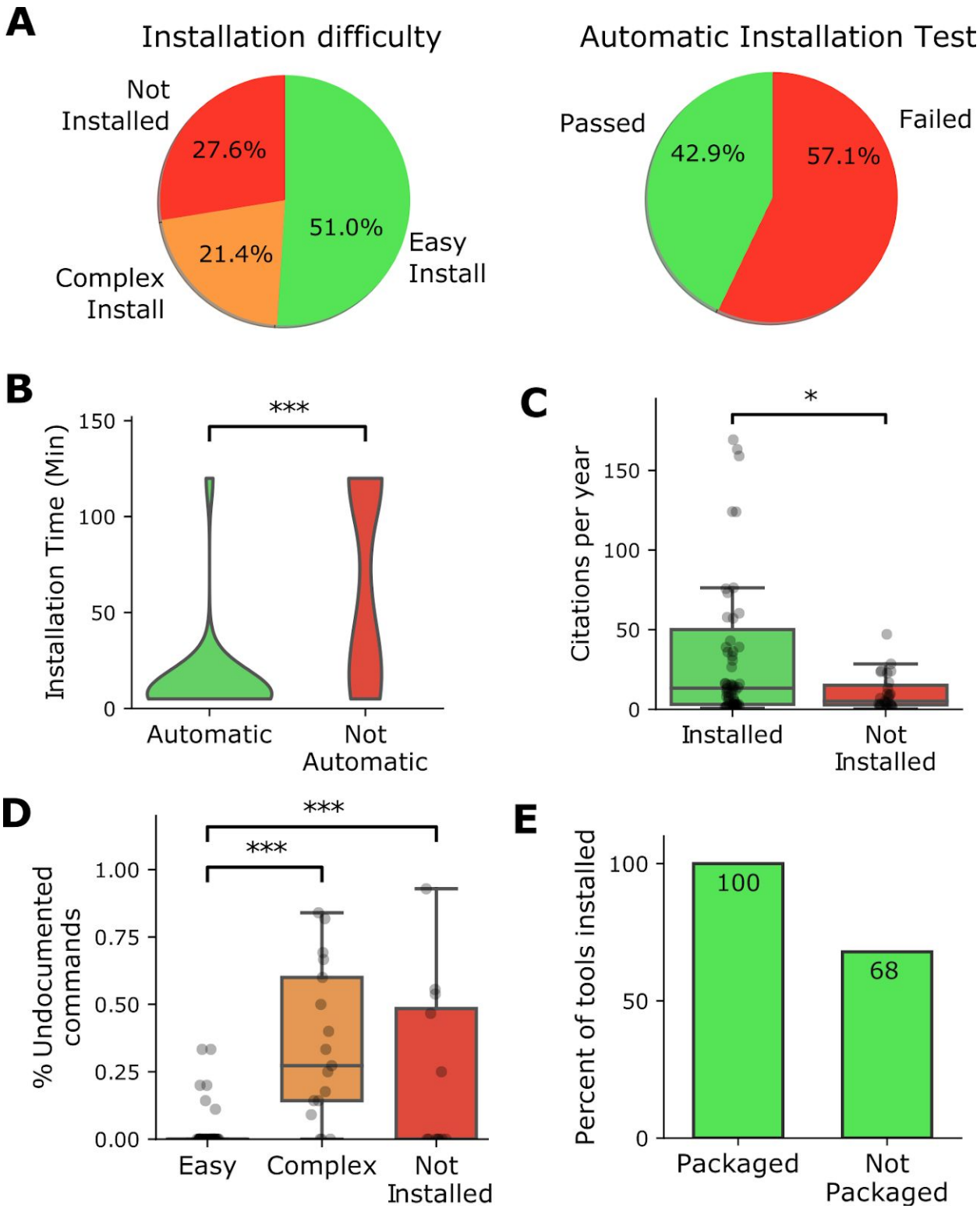

Figure 2. Usability of 99 randomly selected published software tools across 22 life science journals over a span of 15 years. Software tools were categorized as 'easy to install' if the total installation time was 15 minutes or less; 'Complex installation' if the total installation time was longer than 15 minutes but fewer than two hours; 'Not installed' if the software could not be installed in two hours. **(a)** Pie charts showing the percentage of tools with various level of usability. **(b)** Tools that require no manual intervention (pass automatics installation test) exhibit decreased installation time (Kruskal-Wallis, p-value=$4.7x10^{-09}$). **(c)** Tools installed exhibit increased citation per year compared with tools which were not installed (Kruskal-Wallis, *p*-value = 0.032). **(d)** Tools which are easy to install include a decreased portion of undocumented commands (Not Installed vs. Easy Install: Kruskal-Wallis, p-value=$2.3x10^{-6}$, Easy Install vs. Complex Install: Kruskal-Wallis, p-value=$4.7x10^{-6}$). **(e)** Tools available in the well-maintained package managers such as Bioconda were always installable, while tools not shipped via package managers were prone to problems in 32% of the studied cases.

Ideally, all necessary installation instructions should be included in a single script, especially when the number of installation commands is large. In addition, installation scripts should contain commands necessary to install all required dependencies. The vast majority of surveyed tools fail to provide one-line solutions for installation, and, instead, provide step-by-step instructions. On average, eight commands were required to install surveyed tools, while only 3.9 commands were provided in the manual. Among the surveyed software tools, 24 tools provide one-line installation solution, of which nine were available via the Bioconda package manager [33] (Table S1). A package manager is a collection of software tools that

automate the installation, upgrade, and configuration in a consistent manner. Tools with single-command installation require on average six minutes installation time, which is significantly faster when compared to tools which require multi-command installation (Kruskal-Wallis, p-value=4.7×10^{-6}) (**Figure S3**). Tools available in well-maintained package managers (e.g., Bioconda) were always installable, while tools not shipped via package managers were prone to problems in 32% of the studied cases (**Figure 2e**).

Automatic verification of software usability

Software quality, including usability, is typically not thoroughly tested in the formal peer review process. Rather than relying on reviewer feedback, which is often problematic as the reviewers may lack the computational skills and time to verify the tools, it is possible to automate the assessment process when software guarantees access to (i) the software binaries or source code; (ii) a script that installs the software in a given UNIX environment; (iii) a small example dataset and its expected output; and (iv) a script to perform the analysis on the dataset from *(iii)*.

To provide an automated and openly verifiable certification that a tool is usable, we suggest a model of a server that uses public badges to endorse the usability of a software tool. The server will issue a certificate to the software author, which indicates that the proposed software passed an 'Automatic Installation Test.' The installation process, in this case, includes a testing phase that ensures the installation can be successful. Authors of computational tools that submit their software tool to our badge server, alongside an installation script and an example dataset, will receive a badge of confirmation which certifies that the software tool was

successfully installed in a third-party environment. Using a Secure Hash Algorithm [34], each generated badge would be unique to each version of the software, installation script, test dataset, and operating system used by the server.

To validate a badge, the server will use a private cryptographic key to publicly sign the badge. Public badge testing provides a strong endorsement of the tool usability up to the current highest standards in the industry, as only the same software version, installation script, and test dataset will confirm the authenticity of the badge and its public signature. A public badge platform will provide a mechanism for researchers and editors of journals in computational biology to verify the usability of a tool in under five minutes through confirmation of the server's signature. Badges inform the user *a priori* if and under what conditions the software is installable, potentially reducing for each user a significant amount of time that otherwise would be required to test software and attempt installing software that is ultimately uninstallable. To ensure the usability of our badge server, we also provide a small script that automates the verification of badges (see Supplementary Note 2).

In addition to guaranteeing that a software tool can be successfully installed in a standardized environment, the badge also reflects which specific UNIX system was used during the test installation. (UNIX-based systems are the most commonly used operating systems in the field of computational biology.) Furthermore, the badge server does not assume an open source software and can be generated based on the source code or binary files.

Box 1. Principles to increase usability and archival stability of omics computational tools and resources.

The results from our study point to several specific opportunities for establishing an effective software development and distribution practice. Here we present five principles to increase the usability and archival stability of omics computational tools and resources. The majority of surveyed software tools and resources address only a portion of these principles.

1. **Host software and resources on archivally stable services**

   Selecting the appropriate service to host your software and resources is critical. A simple solution is to use web services designed to host source code (e.g. GitHub [36,37] or Sourceforge). In our study, we have determined that more than 98% of software tools and resources stored at GitHub or Sourceforge are accessible, and tools hosted on these services remain stable for longer periods of time (Table S2). Ideally, the repositories storing code should be permanently archived via [38], for example, GitHub or SourceForge releases, Zenodo (https://zenodo.org), or Internet Archive (https://archive.org/).

2. **Provide easy-to-use installation interface**

   Use sustainable and comprehensive software distribution. One example of a sustainable package manager is Bioconda[33], which is language agnostic and available on Linux, UNIX, and Mac operating systems. Bioconda is the most popular package manager, currently covering 2900 software tools that are continuously maintained, updated, and extended

by a growing global community [33]. Bioconda provides a one-line solution for downloading and installing a tool.

3. **Take care of all the dependencies the tool needs**

Even the most widely used tools rely on dependencies. To facilitate simple installation with required dependencies, provide an easy-to-use interface to download and install all dependencies. Package managers can potentially solve this problem since all dependencies are usually preinstalled. Bioconda also automatically generates containers for each Bioconda 'recipe' [39], which provide all files and information needed to install a package. One drawback is that the existing tools in portable package managers are manually updated by the team or community, often delaying such updates. For example, as of August 10, 2018, R 3.5 was unavailable under Bioconda. Alternatively, a simple bash script can be used to combine the commands for installing dependencies and developed software tools into a single script. Forcing users to install such dependencies in a non-configurable location can lead to conflicts. To avoid conflicts, one can design an installation script that installs all dependencies in a user-configurable directory.

4. **Provide an example dataset**

Provide an example dataset inside the software package, with a description of the expected results. Similar to unit and integration testing practices in software engineering, example datasets allow the user to verify that the tool was successfully installed and works properly before running the tool on experimental data. A tool may

be installed with no errors, yet it may still fail to successfully run on the input data. Only 68% of examined tools provide an example dataset (Table S1).

5. Provide a 'Quick Start' guide

Allow the user to verify the installation and performance of the tool. Providing a 'Quick Start' guide is the best way for the user to know that the tools are installed and is working properly. The guide should provide the commands needed to download, install, and run the software tool on the example dataset. An example of a 'Quick Start' guide is provided in Supplemental Note 2. In addition to the 'Quick Start', a detailed manual needs to be provided with information on options and advanced features and configuration of the tool. Best practices of creating bioinformatics software documentation are discussed elsewhere [17].

6. Choose an adequate name

Choose a software name that best reflects the developed tool or resource. Today's "age of Google" places new demands on the function of tool names, which should be memorable and unique, yet easily searchable. In addition, there are no regulations on tool names. For example, there are at least six tools named 'Prism,' making it challenging to find the right tool (Supplementary Note 3). Scout the web to check the uniqueness of a name before publishing a new tool.

7. Assume no root privileges

Tools are often installed on a high-performance computing cluster where users do not have administrative (root/superuser) privileges to install software into system directories. When developing instructions for installation of the proposed software tool, avoid commands that require root access. Examples of such commands include those that use package managers that require root/superuser privileges, such as sudo apt-get install or sudo yum install.

8. Create agnostic installation platform or distribute different versions for each platform

Specification of various versions of UNIX-based systems may limit the usability of software. Developers should either create a tool that will work on any platform or create a separate version for each platform. Platform-specific commands (e.g., Homebrew [40]) should be avoided.

--end Box 1

Discussion

Our study assesses an emergent software crisis in computational biology that is characterized by lack of standards regarding usability and long-term archival stability of omics computational tools and resources. Despite recent requirements on the behalf of journals to impose data and code sharing on published authors' work, 25.0% of 26,631 omics software resources examined in this study are not currently accessible via the original published URLs.

Among the 99 software packages selected for our usability test, 49.0% of computational biology tools failed our 'easy-to-install' test. In addition, 27.6% of surveyed tools could not be installed due to severe problems in the implementation process. One-quarter of examined tools are easy to install and use; in these cases, we identify a set of good practices for software development and dissemination.

Reviewers assessing the papers that present new software tools could begin addressing this problem with the adoption of a rigorous, standardized approach during the peer review process. Feasible solutions for improving the usability and archival stability of peer-reviewed software tools include requirements for providing installation scripts, test data, and functions that allow automatic checks for the plausibility of installing and running the tool. For example, forking is a simple procedure that ensures the version of cited code within an article may persist beyond initial publication [41]. Academic journals recently took a major step toward improving archival stability by permanently forking published software to GitHub (e.g., (Mosqueiro et al. 2017)).

The current workflow of computational biology software development in academia encourages researchers to develop and publish new tools, but this process does not incentivize long-term maintenance of existing tools. Results from this study provide a strong argument for the development of standardized approaches capable of verifying and archiving software. Further, our results suggest that funding agencies should emphasize support for maintenance of existing tools and databases.

Manual interventions and long installation times are unappealing to many users, especially to those with limited computational skills. Many life science and medical researchers

lack formal computational training and may be unable to perform manual interventions (e.g., installing dependencies or editing computer code during installation). Users could leverage advanced knowledge of the time and computational skills required to properly install a software package. We propose a prototype of a badge server that runs an automated installation test, thus introducing to the peer review process explicit assessment of a tool's usability. This badge server would be particularly useful in computational biology, an interdisciplinary field comprised of reviewers who often lack the skills and time to verify the usability of software tools. Many benchmarking studies already routinely report relative ease of installation and use of new tools as components of their performance metrics[44] .

Acknowledgment

We thank John Didion (https://twitter.com/jdidion) for an interesting discussion over Twitter about the issue of software usability.

References

1. Van Noorden, R., Maher, B. & Nuzzo, R. The top 100 papers. *Nature* **514,** 550–553 (2014).
2. Wren, J. D. Bioinformatics programs are 31-fold over-represented among the highest impact scientific papers of the past two decades. *Bioinformatics* **32,** 2686–2691 (2016).
3. Markowetz, F. All biology is computational biology. *PLoS Biol.* **15,** e2002050 (2017).
4. Marx, V. The big challenges of big data. *Nature* **498,** 255–260 (2013).
5. Greene, A. C., Giffin, K. A., Greene, C. S. & Moore, J. H. Adapting bioinformatics curricula

for big data. *Brief. Bioinform.* **17,** 43–50 (2016).

6. Ahn, W.-Y. & Busemeyer, J. R. Challenges and promises for translating computational tools into clinical practice. *Current Opinion in Behavioral Sciences* **11,** 1–7 (2016).
7. Stodden, V., Seiler, J. & Ma, Z. An empirical analysis of journal policy effectiveness for computational reproducibility. *Proc. Natl. Acad. Sci. U. S. A.* **115,** 2584–2589 (2018).
8. Gertler, P., Galiani, S. & Romero, M. How to make replication the norm. *Nature* **554,** 417–419 (2018).
9. Beaulieu-Jones, B. K. & Greene, C. S. Reproducibility of computational workflows is automated using continuous analysis. *Nat. Biotechnol.* **35,** 342–346 (2017).
10. List, M., Ebert, P. & Albrecht, F. Ten Simple Rules for Developing Usable Software in Computational Biology. *PLoS Comput. Biol.* **13,** e1005265 (2017).
11. Baxter, S. M., Day, S. W., Fetrow, J. S. & Reisinger, S. J. Scientific Software Development Is Not an Oxymoron. *PLoS Comput. Biol.* **2,** e87 (2006).
12. Prlić, A. & Procter, J. B. Ten simple rules for the open development of scientific software. *PLoS Comput. Biol.* **8,** e1002802 (2012).
13. Gewaltig, M.-O. & Cannon, R. Current practice in software development for computational neuroscience and how to improve it. *PLoS Comput. Biol.* **10,** e1003376 (2014).
14. Altschul, S. *et al.* The anatomy of successful computational biology software. *Nat. Biotechnol.* **31,** 894–897 (2013).
15. Carpenter, A. E., Kamentsky, L. & Eliceiri, K. W. A call for bioimaging software usability. *Nat. Methods* **9,** 666–670 (2012).
16. Leprevost, F. da V. *et al.* On best practices in the development of bioinformatics software.

Front. Genet. **5,** (2014).

17. Karimzadeh, M. & Hoffman, M. M. Top considerations for creating bioinformatics software documentation. *Brief. Bioinform.* **19,** 693–699 (2018).
18. Queiroz, F., Silva, R., Miller, J., Brockhauser, S. & Fangohr, H. Good Usability Practices in Scientific Software Development. (2017). doi:10.6084/m9.figshare.5331814.v3
19. Jiménez, R. C. *et al.* Four simple recommendations to encourage best practices in research software. *F1000Res.* **6,** (2017).
20. Ősz, Á., Pongor, L. S., Szirmai, D. & Győrffy, B. A snapshot of 3649 Web-based services published between 1994 and 2017 shows a decrease in availability after 2 years. *Brief. Bioinform.* (2017). doi:10.1093/bib/bbx159
21. Support Model Organism Databases. Available at: http://www.genetics-gsa.org/MODsupport. (Accessed: 11th August 2018)
22. Database under maintenance. *Nat. Methods* **13,** 699–699 (2016).
23. Guellec, D. & Van Pottelsberghe De La Potterie, B. The impact of public R&D expenditure on business R&D*. *Economics of Innovation and New Technology* **12,** 225–243 (2003).
24. Ahmed, Z., Zeeshan, S. & Dandekar, T. Developing sustainable software solutions for bioinformatics by the ‘ Butterfly’ paradigm. *F1000Res.* **3,** 71 (2014).
25. Kanitz, A. *et al.* Comparative assessment of methods for the computational inference of transcript isoform abundance from RNA-seq data. *Genome Biol.* **16,** 150 (2015).
26. Chen, S.-S. Digital Preservation: Organizational Commitment, Archival Stability, and Technological Continuity. *Journal of Organizational Computing and Electronic Commerce* **17,** 205–215 (2007).

27. Carnevale, R. J. & Aronsky, D. The life and death of URLs in five biomedical informatics journals. *Int. J. Med. Inform.* **76,** 269–273 (2007).

28. Dellavalle, R. P. *et al.* Information science. Going, going, gone: lost Internet references. *Science* **302,** 787–788 (2003).

29. Ducut, E., Liu, F. & Fontelo, P. An update on Uniform Resource Locator (URL) decay in MEDLINE abstracts and measures for its mitigation. *BMC Med. Inform. Decis. Mak.* **8,** (2008).

30. Wren, J. D., Georgescu, C., Giles, C. B. & Hennessey, J. Use it or lose it: citations predict the continued online availability of published bioinformatics resources. *Nucleic Acids Res.* **45,** 3627–3633 (2017).

31. Wren, J. D. URL decay in MEDLINE--a 4-year follow-up study. *Bioinformatics* **24,** 1381–1385 (2008).

32. Piwowar, H. Altmetrics: Value all research products. *Nature* **493,** 159 (2013).

33. Grüning, B. *et al.* Bioconda: sustainable and comprehensive software distribution for the life sciences. *Nat. Methods* **15,** 475–476 (2018).

34. Dworkin, M. J. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. (2015).

35. Mergel, I. Open collaboration in the public sector: The case of social coding on GitHub. *Gov. Inf. Q.* **32,** 464–472 (2015).

36. Perez-Riverol, Y. *et al.* Ten Simple Rules for Taking Advantage of Git and GitHub. *PLoS Comput. Biol.* **12,** e1004947 (2016).

37. When it comes to reproducible science, Git is code for success. Available at:

https://www.natureindex.com/news-blog/when-it-comes-to-reproducible-science-git-is-code-for-success. (Accessed: 11th August 2018)

38. Thelwall, M. A fair history of the Web? Examining country balance in the Internet Archive. *Libr. Inf. Sci. Res.* **26,** 162–176 (2004).
39. da Veiga Leprevost, F. *et al.* BioContainers: an open-source and community-driven framework for software standardization. *Bioinformatics* **33,** 2580–2582 (2017).
40. Homebrew. *Homebrew* Available at: https://brew.sh/. (Accessed: 17th August 2018)
41. Forking software used in eLife papers to GitHub. *elifesciences.org* (2017).
42. Kaiser, J. BIOMEDICAL RESOURCES. Funding for key data resources in jeopardy. *Science* **351,** 14 (2016).
43. Pickrell, J. beanbag genomics. *beanbag genomics* Available at: https://joepickrell.wordpress.com/. (Accessed: 11th August 2018)
44. Hunt, M., Newbold, C., Berriman, M. & Otto, T. D. A comprehensive evaluation of assembly scaffolding tools. *Genome Biol.* **15,** R42 (2014).
45. Fonseca, N. A., Rung, J., Brazma, A. & Marioni, J. C. Tools for mapping high-throughput sequencing data. *Bioinformatics* **28,** 3169–3177 (2012).
46. Pabinger, S. *et al.* A survey of tools for variant analysis of next-generation genome sequencing data. *Brief. Bioinform.* **15,** 256–278 (2014).

Glossary

Usability. The tool is considered usable if (a) the tool and its corresponding dependencies can be installed on Linux/UNIX-based operating systems, and if (b) the tool can produce expected results from the input data with no errors.

Automated installation test. This test of software installation ease is performed by the biomedical researcher. The researcher is using only installation commands provided in the manual in the recommended order. No extra commands are allowed. A tool passes the automated installation test if the user can successfully install the package following only the commands from the manual.

The package manager is a collection of software tools that automate the installation of a tool's core package and updates in a consistent manner. Package managers help solve the 'dependencies problem' by pre-installing required third-party software packages. Bioconda is one of the most popular package managers for omics computational tools. A growing global community of Bioconda users continuously maintain, update, and extend over 2900 software tools.

Methods

Protocol to check the archival stability of published software tools

We downloaded open access papers via PubMed from 10 systems and computational biology journals from NCBI FTP server (ftp://ftp.ncbi.nlm.nih.gov/pub/pmc/). We included the following journals: *Bioinformatics*, *BMC Genomics*, *Genome Medicine*, *Nature Methods*, *PLoS*

Computational Biology, *BMC Bioinformatics*, *BMC System Biology*, *Genome Biology*, *Nature Biotechnology*, *Nucleic Acids Research*, and *GigaScience*.

Papers were downloaded in XML format containing name-tags for field extraction. (Raw data from PubMed is available at **https://github.com/smangul1/good.software**/.) Specifically, we focused on three name-tags: <abstract>, <body>, and <text-link>. Each paper's abstract is enclosed inside the <abstract> tag (Figure S1). The <body> tag contains the key contents like introduction, methods, results, and discussion. <ext-link> tags contain internet addresses for external sources (e.g., supplementary data and directions for downloading data sources and software packages).

We deployed a heuristic approach in extracting only software links produced by the paper's author(s). We assumed that these links are in <ext-link> tags whose neighbor words contain one of the following keywords: "here", "pipeline", "code", "software", "available", "publicly", "tool", "method", "algorithm", "download", "application", "apply", "package", and "library". The links to the software Biogem are found inside two different <ext-link> tags. The first link, http://www.biogems.info, will be marked by the keyword "available," on the right; whereas the second link, http://www.biogems.info/howto.html, is marked by the same word "available," from the left. The neighborhood for an <ext-link> tab is 75 characters, including spacing (about 5-10 words on average) from the start and end of the tag.

For each extracted link, we used the HTTPError class of the Python library urllib2 to get the HTTP status. Status number 400 and above indicate broken links; for example, the well-known 404 code implies "Page Not Found". We used links found in the body of a paper when the abstract does not contain any external internet links. Since the threshold for the

allotted time may bias the results, we manually verified 1229 URLs reported with the timeout error code (**Figure S1**). Our protocol to check the archival stability of published software tools is freely available at https://github.com/smangul1/good.software.

Protocol to check the usability of published software tools

To standardize the operating system environment for each tool installation, we used a CentOS 7 (v1710.01) Vagrant virtual machine. CentOS is an open-source operating system that is widely used in research computing. To prevent dependency mismatches due to previously installed packages, we installed each tool in a new Vagrant virtual machine. We present a summary of our protocol in Figure S3. Tools were classified into three categories: (1) easy to install, where installation took less than 15 minutes; (2) hard to install, where installation took between 15 minutes and two hours; and (3) not installed, meaning installation took longer than two hours or could not be completed. We tested a total of 99 tools across various categories and fields as described below. Information on the tools tested and the results of the test are available in Table S1.

Tools for microbiome profiling

The usability of 10 common tools for microbiome analysis was tested. To develop a comprehensive list of popular tools, two co-authors co-authors independently made lists of 30 microbiome tools currently used for microbiome data processing, based on a literature survey, and identified those present on both lists. Microbiome tools can vary in their specificity of use;

we limited the final tool list to five tools that process raw sequences into a final OTU table, and five tools capable of broad downstream analysis functions.

Tools for read alignment

We tested the usability of 10 tools for read alignment. We randomly selected a total of 20 tools—10 tools from a recent survey [45] and 10 tools from PubMed (https://www.ncbi.nlm.nih.gov/pubmed/). The full list of extracted URLs is available at https://github.com/smangul1/good.software/wiki . To confirm that the installation process indeed worked, we used reads generated from the complete genome of *Enterobacteria phage lambda* (NC_001416.1).

Tools for variant calling tools

We tested the usability of seven randomly sampled tools designed for variant calling [46]. We confirmed successful software installation when the core functionality of each package could be executed with an example dataset. Only one of the tools was not packaged with an example dataset, in which case we randomly chose an open example dataset. We discarded from our study the tools for which papers could not be located.

Tools for structural variants tools

We examined the usability of 52 common tools used for the structural variant (SV) calling from whole genome sequencing (WGS) data. First, we compiled a list of tools that use read alignment, where reads aligned to the locations are inconsistent with the expected insert size

of the library or expected read depth at a specific locus. We randomly selected 50 tools out of 70 programs designed to detect SVs from WGS data and published after 2011. We confirmed the successful installation of each software package by executing its core functionality with an example dataset.

Additional omics tools

Lastly, we randomly selected 20 published tools based on the URL present in the abstract or the body of the publications available in PubMed (https://www.ncbi.nlm.nih.gov/pubmed/). The full list of extracted URLs is available at https://github.com/smangul1/good.software/wiki .

Statistical analysis

Once the archival information was recorded, variance analysis was performed to assess the differences among the links categorized as 'accessible', 'redirected', 'broken' and 'time out'. We inspected differences in five different statistics: impact factor of the journal where the tool was published; the number of citations in the original paper where the tool was published; number of citations per year in social media platforms such as blogs and twitter feeds; total readership measured by Altmetrics; and the final Altmetric score. Because the distributions of all five measures presented heavy tails and deviated from a bell-shaped distribution, we performed a Kruskal-Wallis test on ranks, followed by a Tukey post-hoc test to confirm which groups presented significant differences with a significance level of 0.01. We provide all

p-values and test statistics from these experiments in our electronic supplemental material on GitHub (https://github.com/smangul1/good.software/wiki/Reproducing-the-results).

Supplementary Notes

Supplemental Note 1. An example of the 'Quick Start'

1. Download the tool using: git clone https://github.com/x/software.tool.git
2. Install tool using: cd software.tool; ./install.sh
3. Run the tool for the example dataset (distributed with the tool): ./software.tool example.dataset

Supplementary Note 2. Badge Server to inspect installation reproducibility.

The server creates an instance of a UNIX virtual machine and runs the installation script and test protocol submitted. If the installation completes without errors, and the test dataset provides the expected result, a badge is created that certifies the usability of the tool under the

tested conditions. The badge consists of a unique summary generated by the Secure Hash Algorithm 3 (SHA-3) [34], of the items submitted by authors. The server then uses a private cryptographic key to publicly sign this summary. There is a small probability that the hashed summary of two different objects created by SHA-3 could be identical (also known as collision); however, this method is the current technological standard of unique badge creation and is broadly accepted by all industries. Using the server's public key and the hashed version of the software, any user can authenticate the signature and prove that the server was indeed able to install the tool without manual intervention.

Researchers would benefit from access to the verification process. We provide a small script that verifies whether a given badge was issued by our server. Verifying the authenticity of the server's signature only takes a few minutes and can be done automatically with this client script. Therefore, our model provides a mechanism for computational biology researchers and journal editors with minimal technical knowledge to verify the usability of a tool in under five minutes. We provide badges endorsing a software package's usability, and both the badge server and the client script are publicly available on GitHub.

Supplementary Note 3

List of bioinformatics tools with name Prism.

- https://www.ncbi.nlm.nih.gov/pubmed/22851530 (Structural Variance)
- https://academic.oup.com/nar/article/43/20/9645/1394603 (Metabolomics)
- https://www.ncbi.nlm.nih.gov/pubmed/21068001 (Viral Genomics)

- http://honig.c2b2.columbia.edu/prism/ (Protein Structure Analysis)

  https://www.ncbi.nlm.nih.gov/pubmed/15991339 (Protein Structure)

- https://www.bits.vib.be/about

Supplementary Tables

tool.ID	package.manager	number.citations.per.year	executed.commands	commands.manual	undoc.commands	Automatic.instalation.test	instalation.time	easy.flag	example.provided
ID1	Other	0.29	7	2	0.7	Fail	5	Not.installed	Y
ID2	Github	0.67	5	3	0.4	Fail	30	Not.installed	Y
ID3	BitBucket	0.71	7	4	0.4	Fail	30	Complex	Y
ID4	Other	1	2	2	0	Pass	15	Easy	Y
ID5	Bioconductor	1	1	1	0	Pass	5	Easy	Y
ID6	Other	1.33	2	2	0	Pass	5	Easy	Y
ID7	Github	1.33	3	3	0	Pass	5	Easy	Y
ID8	Other	1.33	1	1	0	Pass	5	Easy	N
ID9	Other	1.38	N/A	N/A	N/A	N/A	5	Not.installed	N
ID10	Github	1.5	1	1	0	Pass	15	Easy	N
ID11	SourceForge	1.6	5	N/A	N/A	N/A	30	Complex	N
ID12	Github	1.67	1	1	0	Pass	5	Easy	Y
ID13	Other	1.67	3	3	0	Pass	5	Easy	N

ID14	Other	1.71	4	4	0	Pass	30	Not.installed	Y
ID15	SourceForge	1.83	1	1	0	Pass	15	Not.installed	Y
ID16	Other	2	2	2	0	Pass	5	Easy	N
ID17	Github	2.25	12	12	0	Pass	120	Complex	Y
ID18	Other	2.43	2	N/A	N/A	N/A	5	Easy	N
ID19	Other	2.5	7	7	0	Pass	15	Not.installed	Y
ID20	Other	2.6	11	4	0.6	Fail	15	Not.installed	Y
ID21	Other	2.67	3	3	0	Pass	15	Easy	Y
ID22	Github	2.75	7	7	0	Pass	15	Easy	Y
ID23	SourceForge	3	4	4	0	Fail	120	Not.installed	Y
ID24	Other	3	10	10	0	Pass	30	Not.installed	Y
ID25	Other	3	6	6	0	Pass	15	Easy	Y
ID26	Github	3	7	7	0	Pass	15	Easy	N
ID27	Github	3	9	9	0	Pass	5	Easy	Y
ID28	Other	3.2	5	3	0.4	Fail	30	Complex	Y
ID29	Other	3.25	5	4	0.2	Fail	15	Easy	Y
ID30	Github	3.25	1	1	0	Pass	5	Easy	Y
ID31	Other	3.33	N/A	N/A	N/A	Fail	120	Not.installed	Y
ID32	Github	3.33	6	6	0	Fail	120	Not.installed	Y
ID33	Github	3.75	3	2	0.3	Fail	5	Easy	N
ID34	Other	4.33	4	N/A	N/A	N/A	120	Not.installed	N
ID35	Github	4.5	N/A	6	N/A	Fail	120	Not.installed	N
ID36	Other	4.67	1	1	0	Pass	5	Easy	N
ID37	Other	5.11	N/A	N/A	N/A	Fail	120	Not.installed	N

ID38	Bioconda	5.14	1	1	0	Pass	5	Easy	Y
ID39	Github	5.75	4	3	0.3	Fail	120	Complex	N
ID40	Other	6.83	22	4	0.8	Fail	30	Complex	Y
ID41	Other	7	N/A	N/A	N/A	Fail	120	Not.installed	N
ID42	Other	7.17	2	1	0.5	Fail	15	Easy	N
ID43	Other	7.25	N/A	3	N/A	Fail	120	Not.installed	N
ID44	Other	8	4	3	0.3	Fail	15	Easy	Y
ID45	SourceForge	8	1	1	0	Pass	5	Easy	N
ID46	Other	8.9	14	14	0	Fail	120	Not.installed	Y
ID47	Other	9.56	N/A	N/A	N/A	Fail	120	Not.installed	Y
ID48	Bioconda	9.71	1	1	0	Pass	15	Easy	N
ID49	Other	9.8	N/A	1	N/A	Fail	120	Not.installed	Y
ID50	Other	10.43	N/A	N/A	N/A	N/A	120	Complex	Y
ID51	SourceForge	10.56	N/A	N/A	N/A	N/A	5	Easy	N
ID52	Other	12.17	1	1	0	Pass	5	Easy	Y
ID53	SourceForge	12.25	N/A	N/A	N/A	N/A	5	Easy	N
ID54	Other	12.75	4	4	0	Pass	15	Easy	Y
ID55	SourceForge	13.29	1	1	0	Pass	120	Not.installed	N
ID56	Other	13.33	5	5	0	Pass	15	Easy	Y
ID57	Github	13.75	5	N/A	N/A	N/A	30	Complex	N
ID58	Bioconda	13.83	1	1	0	Pass	5	Easy	Y
ID59	Other	14.7	13	13	0	Pass	15	Easy	Y
ID60	Other	15.1	4	2	0.5	Fail	120	Complex	Y

ID61	Other	15.3	1	1	0	Pass	5	Easy	Y
ID62	Other	15.63	52	N/A	N/A	N/A	120	Complex	N
ID63	SourceForge	16	2	N/A	N/A	N/A	5	Easy	N
ID64	Github	16.25	8	6	0.3	Fail	60	Complex	Y
ID65	Other	16.29	1	1	0	Pass	5	Easy	Y
ID66	Github	16.89	10	3	0.7	Fail	120	Not.installed	Y
ID67	SourceForge	22.71	190	7	1	Fail	120	Not.installed	Y
ID68	Other	23.6	N/A	N/A	N/A	N/A	30	Not.installed	Y
ID69	Other	24	4	N/A	N/A	Fail	120	Not.installed	N
ID70	Other	24.2	N/A	7	N/A	Fail	120	Not.installed	Y
ID71	Bioconductor	26.57	2	1	0.5	Fail	5	Easy	N
ID72	Other	28.5	N/A	29	N/A	Fail	120	Not.installed	N
ID73	Other	30.75	7	3	0.6	Fail	120	Complex	Y
ID74	Other	33.45	3	3	0	Pass	5	Easy	Y
ID75	Bioconductor	36	23	2	0.9	Fail	120	Complex	Y
ID76	BitBucket	36.25	30	10	0.7	Fail	120	Complex	Y
ID77	Other	39	3	3	0	Pass	5	Easy	Y
ID78	Other	39.1	12	12	0	Pass	15	Easy	Y
ID79	Other	43	5	1	0.8	Fail	120	Complex	Y
ID80	Other	47.13	N/A	2	N/A	N/A	5	Not.installed	N
ID81	Bioconda	57.2	1	1	0	Pass	5	Easy	Y
ID82	Bioconductor	57.83	2	2	0	Pass	5	Easy	Y
ID84	Bioconda	60.3	1	1	0	Pass	5	Easy	Y

ID85	pip	73.17	12	10	0.2	Fail	60	Complex	Y
ID86	Github	75.71	1	1	0	Pass	5	Easy	N
ID87	SourceForge	76.3	1	1	0	Pass	5	Easy	Y
ID88	Bioconda	124	1	1	0	Pass	5	Easy	Y
ID89	Other	124.1	1	1	0	Pass	5	Easy	N
ID90	Bioconductor	159	20	2	0.9	Fail	60	Complex	Y
ID91	Bioconda	163.2	1	1	0	Pass	5	Easy	Y
ID92	Other	169.22	2	2	0	Pass	5	Easy	Y
ID93	Github	216.33	12	3	0.8	Fail	30	Complex	Y
ID94	Bioconda	247.4	1	1	0	Pass	5	Easy	Y
ID95	Other	683.11	2	2	0	Pass	5	Easy	Y
ID96	Github	798.4	3	2	0.3	Fail	5	Easy	Y
ID97	Other	1059	10	7	0.3	Fail	60	Complex	Y
ID98	Bioconda	1122.6	1	1	0	Pass	5	Easy	Y
ID99	Bioconda	1450.9	3	3	0	Pass	5	Complex	Y

Table S1. Accessibility and usability of 99 published software tools over 2004-2018 period.

Link location	Journal name	pubmed id	Year	Link	Url status
abstract	BMC_Bioinformatics	12150718	2002	http://sourceforge.net/projects/slritools/	301
abstract	BMC_Bioinformatics	12401134	2002	http://sourceforge.net/projects/slritools/	301
body	BMC_Bioinform	1188225	2002	http://tacg.sourceforge.net	200

	atics	0			
body	BMC_Bioinformatics	12019022	2002	http://www.sourceforge.net/projects/slritools/	301
body	BMC_Bioinformatics	12150718	2002	http://sourceforge.net/projects/slritools	301
body	BMC_Bioinformatics	12493080	2002	http://squirrel-sql.sourceforge.net/	200

Link location	Journal name	pubmed id	Year	Link	Url status
body	BMC_Bioinformatics	19732427	2009	http://github.com/egonw/xws-taverna/tree/master	301
body	Bioinformatics	19417059	2009	http://github.com/semin/ulla	301

Table S2. List of earliest published software tools and resources stored on http://sourceforge.net and https://github.com/

Supplementary Figures

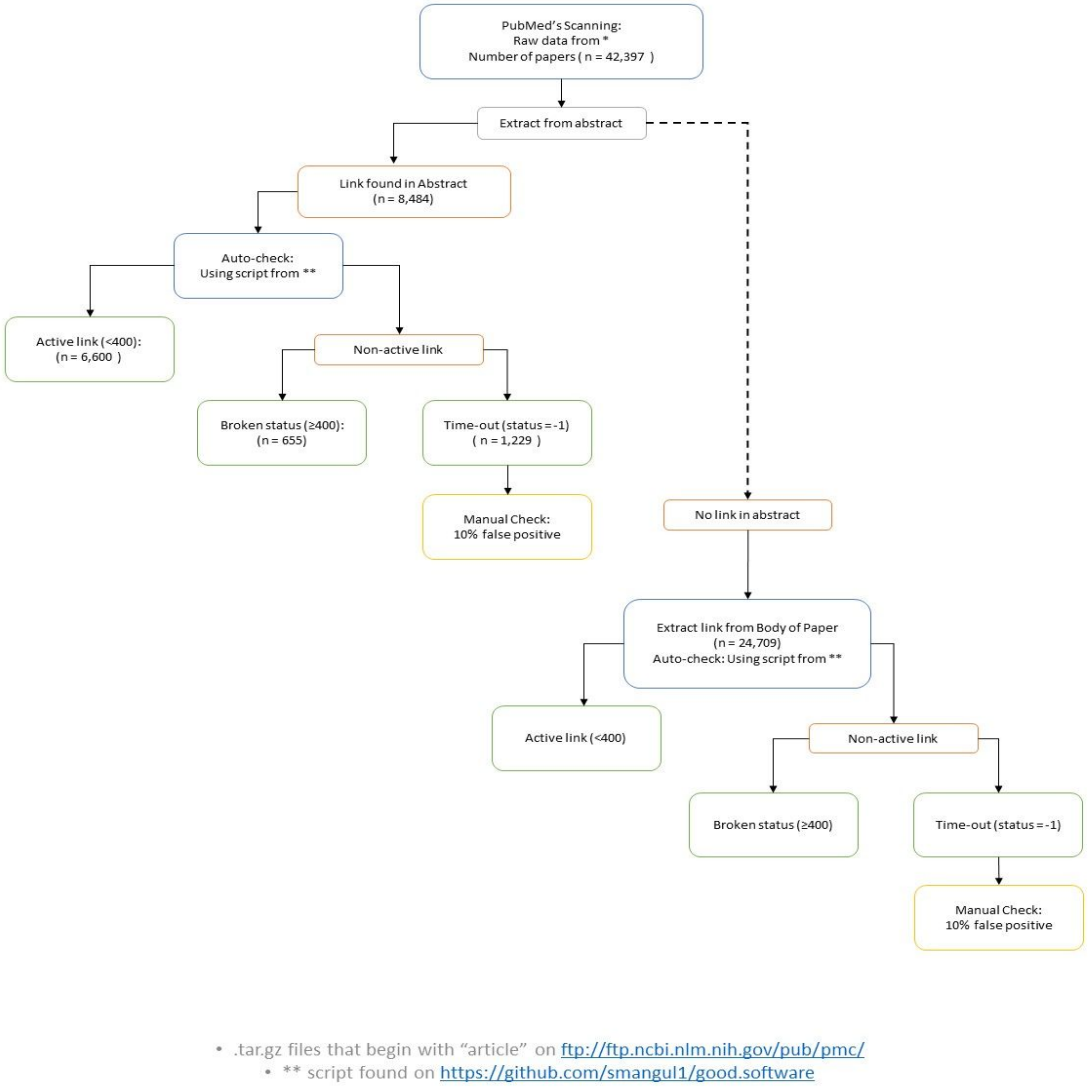

Figure S1. Protocol to check the archival stability of a published software tool or resource. Numbers are provided for illustrative purposes and correspond to the link presented in the abstracts of the published papers considered in this study.

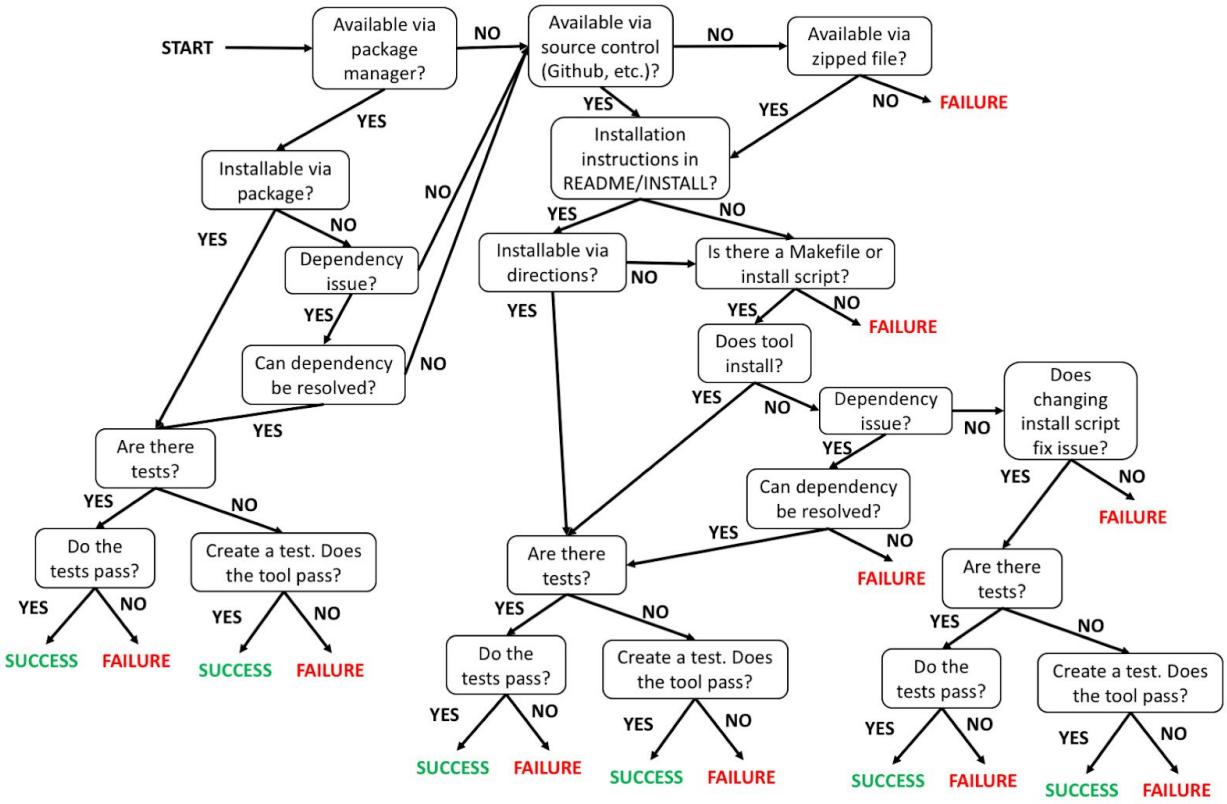

Figure S2. Protocol to verify the usability of a published software tool.

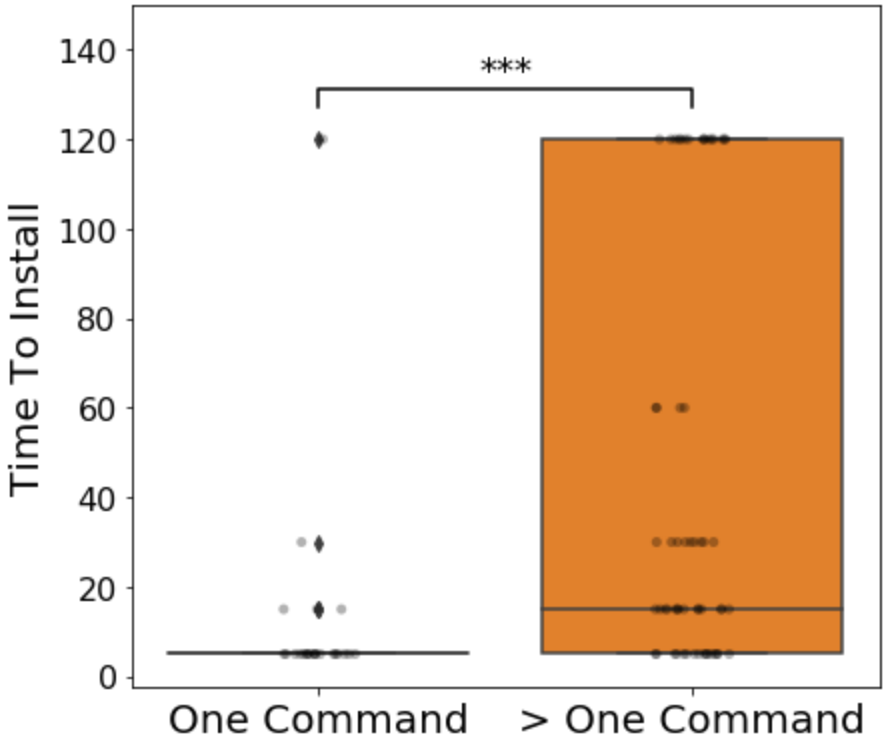

Figure S3. Effect of the number of commands used to run a published software tool on the installation time (Kruskal-Wallis, p-value=4.7×10^{-6})