analytical chemistry

pubs.acs.org/ac

Perspective

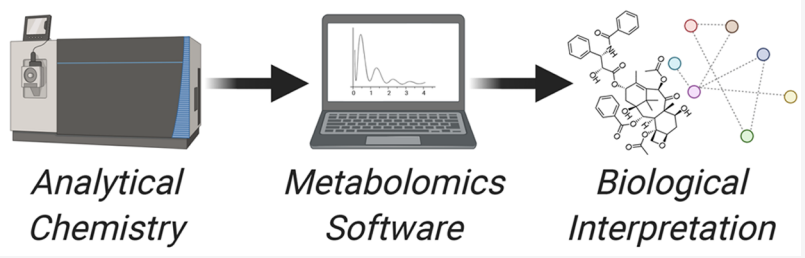
A Practical Guide to Metabolomics Software Development

Hui-Yin Chang, Sean M. Colby, Xiuxia Du, Javier D. Gomez, Maximilian J. Helf, Katerina Kechris, Christine R. Kirkpatrick, Shuzhao Li, Gary J. Patti, Ryan S. Renslow, Shankar Subramaniam, Mukesh Verma, Jianguo Xia, and Jamey D. Young*

Cite This: *Anal. Chem.* 2021, 93, 1912–1923

Read Online

ACCESS | Metrics & More | Article Recommendations | Supporting Information



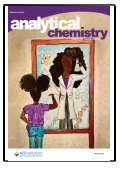
ABSTRACT: A growing number of software tools have been developed for metabolomics data processing and analysis. Many new tools are contributed by metabolomics practitioners who have limited prior experience with software development, and the tools are subsequently implemented by users with expertise that ranges from basic point-and-click data analysis to advanced coding. This Perspective is intended to introduce metabolomics software users and developers to important considerations that determine the overall impact of a publicly available tool within the scientific community. The recommendations reflect the collective experience of an NIH-sponsored Metabolomics Consortium working group that was formed with the goal of researching guidelines and best practices for metabolomics tool development. The recommendations are aimed at metabolomics researchers with little formal background in programming and are organized into three stages: (i) preparation, (ii) tool development, and (iii) distribution and maintenance.

Over the past decade, the field of metabolomics has been revolutionized by advances in chromatography, mass spectrometry (MS), nuclear magnetic resonance (NMR), and many other analytical technologies. These advances have enabled researchers to not only identify and measure a broad diversity of cellular metabolites but also assess changes in metabolic pathway activity or flux. The growth in instrument capabilities has spurred a concurrent proliferation of software tools for processing and analyzing metabolomics datasets and for predicting chemical properties measured by these instruments. In order to convert the vast amounts of data produced by modern metabolomics experiments into interpretable results, analytical chemists often face the necessity of automating data processing workflows or supervising programmers and data analysts who perform software development tasks. For most metabolomics projects, measurement acquisition and data analysis go hand-in-hand, and software needs should be assessed at an early stage of experimental design to ensure that the raw data can be appropriately processed and interpreted. Furthermore, the format of the raw data and the type of analytical instruments have major impacts on how the data should be analyzed.

General data processing tools are typically provided by instrument manufacturers and are intended to meet a wide range of customer needs within a user-friendly package. However, there are many advanced metabolomics applications that demand software solutions contributed by the scientific research community. These tools may address a niche area that is beyond the scope of vendor software, or they may enable entirely new metabolomics approaches to be developed and tested. Furthermore, it is often necessary to combine a sequence of tools together into innovative metabolomics workflows, which requires understanding of the ontology of tools and their usage requirements. For these reasons, analytical chemists and other metabolomics researchers are intimately involved in the process of selecting, creating, extending, and applying software tools according to their research needs.

A common reality is that most metabolomics researchers lack formal training in computer science or software development. Consequently, numerous pitfalls can arise from the lack of knowledge about good software practices. For example,

Received: August 23, 2020
Accepted: December 30, 2020
Published: January 19, 2021



inappropriate code revision could lead to inaccurate results when the code is used by other scientists. Poor user interface design could hinder adoption of the software by non-specialists. Lack of proper documentation can render a codebase impossible to update or even use effectively once the developers transition to other projects. This Perspective is intended to provide a practical summary of considerations that arise at multiple stages of the software development process (Figure 1), aimed at metabolomics researchers with little

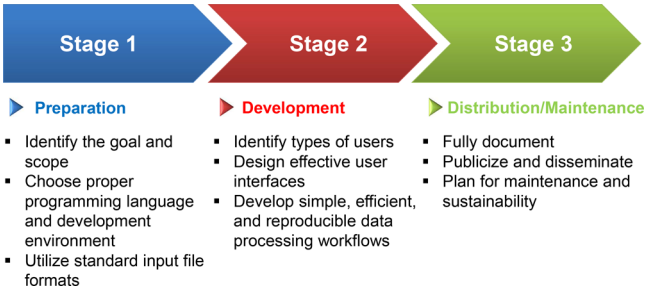


Figure 1. Recommended guidelines at each stage of the software development pipeline.

formal background in programming. The experience from several metabolomics-oriented software development projects has been compiled to help beginners avoid some of the more common mistakes. Rather than presenting a list of non-negotiable rules, we provide essential background information and recommended guidelines to facilitate the software development process and enhance the impact of the final product. Much of the discussion is centered around mass spectrometry due to its wide use in metabolomics studies and the prevailing expertise of the authors, but we expect that the principles discussed can be readily generalized to other measurement technologies as well. This is not intended to be an endorsement of a specific platform or to exclude other technologies, but simply reflects our inability to be fully comprehensive in a paper of this length and scope.

STAGE 1: PREPARATION

Defining crucial aspects of the project before starting to write code can help avoid future shortcomings and delays in tool development. The most important points to consider during the pre-development stage are described in the following sections.

Identify the Goal and Scope of the Tool. Software tools for metabolomics studies generally fall into one of six categories based on their functions:

Data pre-processing software automates the picking, deconvolution, matching, and alignment of signals (i.e., peaks) present in the raw data.[1] For MS data pre-processing, this workflow can also include operations such as baseline correction, noise reduction by smoothing and filtering, or de-isotoping the peaks.

Molecular structure identification software (also called annotation software) matches the processed peaks with databases in order to attribute evidence for the presence of a specific molecular compound or to directly predict a chemical structure (i.e., using electron diffraction or NMR measurements). The databases could be public or private repositories derived from analysis of authentic chemical standards, or from *in silico* property predictions. These tools are central to analyzing untargeted metabolomics datasets, as a pre-defined list of target compounds is not required prior to searching for potential database "hits".

Statistical analysis software compares data from multiple samples in order to identify important features of the dataset that vary significantly in an experiment. For univariate analysis, basic methods such as analysis of variance (ANOVA) or *t*-tests can be implemented. However, most metabolomics experiments require advanced multivariate statistical methods that account for multiple experimental factors and can analyze datasets with more features (i.e., variables) than samples. Principal components analysis (PCA) is an unsupervised method often used as a first approach to characterize the variation in metabolomics datasets. On the other hand, supervised methods such as partial least squares discriminant analysis (PLS-DA) and orthogonal partial least squares (OPLS) can be used to identify features of the data that distinguish experimental groups. Other supervised and unsupervised machine learning methods (e.g., clustering, linear discriminant analysis, random forest, neural networks) are oftentimes implemented in statistical analysis software tools. Additional data processing steps may be required prior to statistical analysis: e.g., batch correction, normalization, filtering, and imputation of missing values.

Functional analysis software generally uses information on the annotated peaks and their corresponding properties (i.e., areas, widths, isotope ratios, etc.) to infer changes at different functional levels such as metabolic reaction modules, pathways, or sub-networks to facilitate biological interpretation. The most common approach involves analyzing peak intensities from multiple samples to determine which metabolic pathways are altered in an experiment. However, the "function" of a metabolomic marker can also be associated with a disease, phenotype, molecular mechanism, or class of metabolites rather than a pathway.[2] Usually functional analysis requires a list of known or inferred metabolites to integrate into the species- and context-dependent metabolic network of the biological system under investigation. In many cases, this involves reconstruction of the network based on prior knowledge of enzymes, reactions, and pathways using automated text mining or manual curation.

Chemical property prediction software is designed to build or augment libraries of chemical properties that can be measured in metabolomics experiments (i.e., relative intensities, m/z, retention times, collision cross sections, chemical shifts, MS/MS spectra, isotopic signatures, etc.). Libraries can consist of properties derived from laboratory analysis of authentic reference standards (the traditional gold standard approach) or from the use of property prediction software to create *in silico* libraries, which can be applied to "standards-free" methods of identification.[3,4] Historically, predicted m/z values have been routinely used in MS-based metabolomics analysis, as this property is readily calculated at higher accuracy than is achievable experimentally. Recently, software based on molecular dynamics, quantum chemical calculations, and deep learning has been used to predict NMR chemical shifts, ion-mobility collision cross sections, chromatographic retention times, MS fragmentation patterns, adducts or isotopic signatures, and other chemical properties.

Metabolic modeling software facilitates the development of kinetic or flux models of metabolic networks. These tools convert lists of enzymes or biochemical reactions into a system of coupled mass balance equations, which are augmented with

https://dx.doi.org/10.1021/acs.analchem.0c03581
Anal. Chem. 2021, 93, 1912−1923

appropriate kinetic and thermodynamic parameters to convert into a mathematical model. The models provide a dynamical assessment of metabolic networks and enable prediction or estimation of metabolic fluxes from metabolomics measurements.[5−7]

To avoid the creation of redundant software that duplicates the base functionality of other domain- or data-specific tools, developers should first familiarize themselves with existing software before "reinventing the wheel". Several compilations of available software tools within each of the aforementioned categories have been published and can be consulted for further information.[8−10] Naturally, existing tools may not support a new data type or implement the latest breakthrough algorithm. In this case, extending a previously developed tool by adding a new module or companion program may be the best way to move a project forward while avoiding unnecessary effort and minimizing adoption barriers. In most cases, it is more efficient to create a module that can be integrated with existing metabolomics workflows, rather than recreating the entire workflow within a single piece of software (Figure 2).

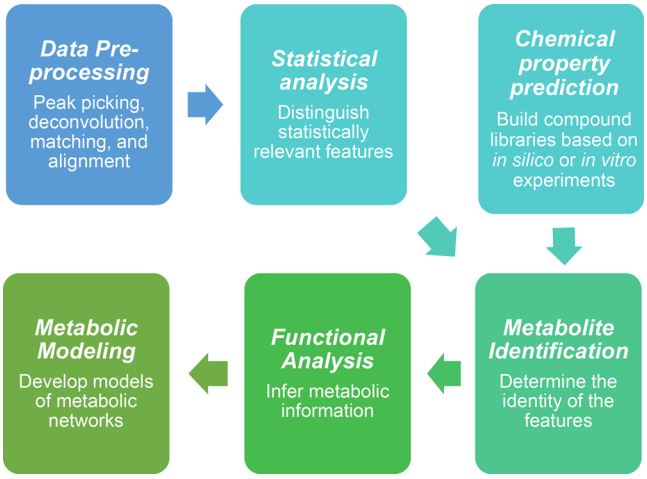


Figure 2. Overview of a typical metabolomics workflow for analysis of MS datasets.

However, adding functionality to software developed by another group often requires a collaborative effort and necessitates planning and mutual trust between the groups to ensure fair attribution. If it is not feasible to build onto an existing tool, e.g., because the existing software is no longer actively developed, is not open-source, or uses an obscure programming language, it may be necessary to implement (or refactor) the desired functionality into a new software package. In this case, it is important to precisely define what elements are missing from the currently available tools and which input/output formats allow for optimal data exchange with existing software and databases.

Choose an Appropriate Programming Language. Before selecting a programming language, an important consideration is that software maintenance and integration can be significantly more costly than the initial software development. If a project is intended for continuation beyond 3−5 years, the codebase should be limited to mainstream programming languages. If the software will be integrated with other software or frameworks, the technology of those companion projects should also be considered. For a small, standalone project to address a short-term research problem, it is acceptable to experiment with new languages and tools. However, for any long-term project, a developer must consider how the software can be maintained and updated while adapting to personnel turnover, available expertise, and change of technologies over time.

The most popular general-purpose programming languages include C-family languages (Java, C#, PHP, C++, C) and Python, as corroborated by the recent Stack Overflow survey.[11] The platform independence of Java is a key advantage over other C-family languages, but it comes at the expense of slower performance. The popular network visualization tool Cytoscape,[12] the MS/MS analysis software SIRIUS,[13] and the general-purpose metabolomics program MZmine 2[14] are all written in Java. Other scientific or general-purpose programming languages may be preferred for certain applications, based on their desktop graphical user interface (GUI) frameworks, specialized libraries or built-in functions (e.g., MATLAB toolboxes), or the available skill set of the developer. For many scientific applications, including bioinformatics and data science, the most popular languages are Python and R.[15] Both are high-level, interpreted languages that prioritize the developer's productivity. While use of these languages may raise concerns over efficiency of execution, the performance penalty is negligible for many applications since both Python and R utilize low-level function libraries that are implemented in C or Fortran, which provide optimized runtime performance for critical steps.

R is a programming environment dedicated for statistical computing and graphics that offers several built-in and publicly available packages covering a wide variety of statistical analyses. Although it has a less conventional syntax that may feel unusual to users with experience in a general-purpose programming language, this has not stopped it from occupying an important niche in areas like functional genomics and metabolomics. The differences between Python and R are also reflected in the available libraries for both languages. Python has many numerical libraries such as Numpy, SciPy, Pandas, and TensorFlow, while R facilitates flexible plotting and graphics, is a leading resource for statistical analyses, and can be extended with many advanced bioinformatics packages. In particular, the Bioconductor R repository provides a wide range of cross-compatible bioinformatics tools (e.g., XCMS[16]).

Python is the fastest-growing major programming language, is ubiquitous across the sciences and in industry (especially deep learning applications), is intuitive and approachable, and, thus, is widely taught in computer science classes. It is the basis of the CoreMS[17] mass spectrometry framework and many other public metabolomics software projects. Python has a very large and active community in the sciences, which has only grown in recent years. The Software Carpentry[18] initiative has created and maintains reusable courseware for teaching and learning Python and R. The rich ecosystem of Python makes it a top choice for projects involving scientific computing, as it can be easier to port between desktop/laptop, high-performance computing, and cloud computing resources than other languages. Ultimately, however, the best choice of programming language will depend on the needs of users, the expertise of the development team, and the intended use/functionality of the tool.

Choose the Software Development Environment. The programming environment for software development includes code editors, version control software, and testing tools. Many excellent code editors are available, including free

https://dx.doi.org/10.1021/acs.analchem.0c03581
Anal. Chem. 2021, 93, 1912−1923

Table 1. Standard Data Formats Supported by Widely Used Metabolomics Software[a]

software	category	import	export
MetaboAnalyst[22]	general purpose	NetCDF, mzXML, mzDATA, .csv, .txt, mzTab	.SVG, .png, .csv, .txt, report (.pdf)
MZmine 2[14]	general purpose	Vendor formats, mzML, mzXML, NetCDF, mzTab	.csv, mzTab, XML, SQL, .msp
MS-DIAL[26]	preprocessing	.ABF	.mgf, .msp, .txt, .mat
XCMS[27]	preprocessing	NetCDF, mzXML, mzDATA	.csv
GNPS[28]	statistics/pathway analysis/metabolite identification	mzXML, mzML, mgf	.graphML, .csv
XCMS online[21]	general purpose	NetCDF, mzXML, mxData	.csv
Mummichog[29]	pathway analysis	.txt, .tsv	.xlsx, .csv
Chemical Property Prediction Tools	cheminformatics/molecular modeling/quantum chemistry	SMILES, InChI, .xyz, .mol2, .sdf, .pdb	SMILES, InChI, .xyz, .mol2, .sdf, .pdb
MS PepSearch[30]	metabolite identification	.msp, .mgf	.csv

[a]This table does not include proprietary software formats, data formats used to save intermediate results of analysis sessions, or raw MS data containers.

ones such as Atom, Sublime, Vim, and VS Code. Version control is an integral part of software development, and Git[19] has become the most common choice today. Older systems such as Subversion (SVN) or Concurrent Versions System (CVS) may also still be in use by some groups. Version control systems are highly recommended when working in teams and/or for the long-term maintenance of software.

The combination of a code editor and a full stack of tools—compiler/interpreter, project management, and version control—is often referred to as an integrated development environment (IDE). Famous examples of IDEs include Visual Studio, NetBeans, IntelliJ IDEA, PyCharm, Spyder, RStudio, and Eclipse, which enable developers to test the code regularly and easily. For web development, a local test server is usually needed, and it is now included in many IDEs. For desktop applications that use a platform-specific runtime environment or compiler, testing under different operating systems (OSs) is a major undertaking. Although Microsoft Windows, Mac OS, and Linux are the major OSs, each has many different versions and/or distributions. Developers often turn to virtual machines to test programs on different OSs or use online services for continuous integration that automatically test software every time new code is committed to the online repository. Cloud technologies like Kubernetes, Docker, and Singularity containers have become very popular since they allow packaging of dependencies with the software tool itself in order to avoid incompatibilities when deployed under different OSs.[20] As a result, containers also become a part of the local development environment, especially for cloud applications.

Assess Needs for Cloud Computing and Virtualization. Cloud computing has become increasingly versatile and offers transparent and portable use of workflows, protocols, and datasets. XCMS Online[21] and MetaboAnalyst[22] have already migrated to cloud servers for improved performance and accessibility. Aside from enabling portability, cloud technologies force a developer to document dependencies and settings needed to run a software stack. This can be particularly important where a tool relies on a specific database (version) or where a tool is built as a helper application extending another open-source tool. The cloud runs on instances, a concept similar to that of a virtual machine. However, cloud instances are assembled on the fly and run off a shared infrastructure. True cloud platforms, such as those hosted by Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, and the open-source OpenStack, make use of containers for deploying cloud infrastructure. A container is a virtual runtime environment that emulates the OS and other program dependencies needed to execute a particular software tool.

The most popular container platform is Docker. A Docker container is an execution environment that uses a "Docker file" to assemble software together with its dependencies to create a unique environment that can be reproduced easily on any computer. MetaboAnalyst, NWChem,[23] and other tools are available as a Docker image, which enables them to be downloaded and installed locally on a lab server or cluster. Singularity is another important platform, especially as it contains several security features that allow for its use under compliance regimes, such as may be required in classified research. Kubernetes is an orchestration framework, often discussed in companion with Docker, which enables cloud instances to be programmatically started or "spun up" and closed. One reason Docker is popular is because of Docker Hub, a free and open way to share Docker containers. When popular software is released, it is common for software publishers to include updated Docker containers on Docker Hub. There are several tutorials for creating Docker files. The PhenoMeNal[24] project has curated a set of tutorials with metabolomics software developers in mind.

For cloud applications, Jupyter[25] notebooks are used routinely to ensure all processes and outputs are available seamlessly to the user community. A Jupyter notebook is a web application that blends different data such as text, live code and its output, data visualization, and multimedia in a single computable document. Its main components include the web application (web server) that creates and edits notebook documents, the kernel that executes the code in the document, and the web browser through which the user interacts with the notebook. The web server can be run on a local host or a remote server. This simple yet powerful architecture, combined with open-source software and the ability to share notebooks, promotes collaboration and increased productivity among research groups. The functionality of notebooks can be enhanced with an ever-expanding list of add-ons developed by the community.

Plan to Use Standard Input/Output File Formats. Many metabolomics software tools have import and export options for a variety of standard data formats (Table 1). Building support for these existing formats is recommended to enable facile exchange of data between different tools and to

https://dx.doi.org/10.1021/acs.analchem.0c03581
Anal. Chem. 2021, 93, 1912−1923

avoid fragmentation of metabolomics data into many incompatible file formats. It also enables files to be readily transferred to/from metabolomics data repositories such as MassIVE,[31] Metabolomics Workbench,[32] or MetaboLights.[33] The choice of data exchange format depends on the type of data and what other programs should be able to read the files. Some of the most popular file types for analysis of metabolomics data are detailed in the Supporting Information. In general, open data formats have become indispensable for data exchange and analysis in the metabolomics community. However, they have weaknesses as well. First, they do not capture raw data in its entirety, and therefore it is important to keep a copy of the raw data in each instrument's proprietary format. Second, they are not specifically designed for efficient computation and storage. Despite these weaknesses, it is still strongly encouraged that metabolomics software tools be designed to read and write open data formats. In particular, mzML and mzTab, as the *de facto* standard open data formats, should be supported for most MS data analysis tools.

STAGE 2: DEVELOPMENT

Metabolites isolated from a biological sample commonly produce tens of thousands of "features" when analyzed by LC/MS, where a feature is defined as a measured ion having a unique pair of retention time and m/z values. Generally, there are three approaches that an investigator can apply to analyze the results: targeted data processing, untargeted data processing with pre-configured software, and advanced data processing with customized software. Each requires a different level of user expertise, as briefly described below. Based on these distinctions, we explore the elements that should be implemented in a metabolomics software tool and their respective design considerations. We note that this classification of users is arbitrary and used only to guide our discussion.

Characteristics of Basic Users of Targeted Software. We consider targeted software users to be those who are only interested in extracting information about a small number of metabolites (e.g., TCA cycle intermediates), even though their datasets may be more comprehensive in nature. This level of analysis can typically be performed by users with pre-packaged software and limited coding experience. For targeted analysis of MS data, the user typically inputs an m/z value of interest and an error window over which it should be monitored. The software then creates an extracted ion chromatogram, which is a plot of the specified ion current as a function of time. For quantitation, the extracted ion chromatogram can then be integrated automatically by the software. Depending on the program, each MS dataset may need to be processed separately. It is feasible to perform such analysis for a limited number of metabolites. However, for global processing of the data in an untargeted manner, these simple manual-processing steps become impractical.

Desirable program features for targeted users include capabilities that suggest input parameters to facilitate the rapid processing of individual data files based on previously analyzed samples or experiments. Additionally, options extending integration bounds that are manually input by a user for a single run to the entire dataset are important for quantitative reliability and speed when a large number of samples are to be analyzed. Given that targeted users are not interested in comprehensive profiling, the considerations described below are not applicable to these researchers.

Characteristics of Basic Users of Untargeted Software. This group of users is interested in processing metabolomics data in a more comprehensive manner, but they do not have extensive experience in bioinformatics or coding. It is important to emphasize that for untargeted metabolomics, the processing workflow described above is impractical because thousands of features would have to be individually inspected. Thus, some MS vendors offer software for the global processing of metabolomics data to support untargeted workflows. A limitation is that vendor solutions are only compatible with data generated from a specific manufacturer's instrument. Notably, vendor-agnostic "plug-and-play" software platforms have also been created in recent years. Some of these solutions are freely available (e.g., XCMS Online, MetaboAnalyst), while others are commercial (e.g., Progenesis). A common theme of these programs is that, in principle, an understanding of the underlying algorithms is not required. Moreover, given that each solution has its own graphical user interface, no expertise in computer programming is needed to use the tools.

At a high level, the "plug-and-play" options perform three basic functions: (i) feature detection, (ii) feature alignment, and (iii) statistical analysis. In brief, this allows the average intensity of a feature to be compared between sample groups. A complication is that performance is parameter-dependent. It is common for users of these platforms to have a limited understanding of the parameters, which can lead to improper processing of the data files. Additionally, like any area of scientific research, there are opportunities to improve the algorithms. Even when optimal parameters are selected, each platform is susceptible to processing artifacts caused by challenges like establishing consistent baselines or integration bounds. Finally, the functionalities of these software platforms are mostly limited to simple experimental designs such as comparisons of one or more experimental groups with a control group. Automated metabolite identification and advanced experimental designs, like the modeling of isotope labeling patterns, kinetic time courses, or dose−response curves, are generally not supported.[34]

Characteristics of Advanced Users. We consider advanced users to be comfortable with coding and/or developing their own custom metabolomics workflows versus using pre-packaged software. The overwhelming majority of algorithms that have been developed for analysis of untargeted metabolomics data are only available with a command-line interface and custom workflows, which require more advanced programming skills.[10] In part, this is because creating an intuitive GUI requires a substantial investment of resources. Typically, only well-established algorithms that are widely used by the community evolve into such GUI-based platforms. It has become increasingly common for research laboratories performing untargeted metabolomics to have at least one investigator who is proficient in computer programming languages (R, C, Python, MATLAB, etc.) to investigate new algorithms or for dealing with new instrument data types. At the most basic level, these researchers allow laboratories to apply metabolomics programs that would otherwise be inaccessible. This could be particularly useful to investigators interested in using the newest processing methods or to laboratories with specialized data-processing needs that are not satisfied with the standard plug-and-play options.

Advanced users typically require custom solutions with modified algorithms and advanced processing functionalities to

https://dx.doi.org/10.1021/acs.analchem.0c03581
Anal. Chem. 2021, 93, 1912−1923

support experimental designs beyond simple pairwise comparisons. They can improve upon existing programs by modifying code, or they can assemble entirely new workflows by chaining existing modules together in sequence. It is our perspective, however, that there are two general levels of researchers with competence in computer programming: users and coders. Users are able to run existing programs in languages like Python and R, but do not have an extensive understanding of the functions being implemented. Coders, on the other hand, have a working knowledge of the underlying algorithms and can modify functions as needed. When creating new programs, we recommend that developers recognize that much of the target audience is likely to be users rather than coders and thoroughly document their code and create user manuals (or additionally, provide working examples with toy data), as described in Stage 3.

Provide Graphical and/or Interactive User Interfaces Appropriate for Target Users. The user interface (UI) is an important bridge between users and software that allows users to control program execution.[35,36] The goal of a UI design is to provide an easy, efficient, and friendly environment for software operation. Both GUIs and command-line interfaces (CLIs) are commonly used for bioinformatics software. For example, MZmine,[14] MS-FINDER,[37] and iMet-Q[38] are tools with GUIs, and XCMS,[27] CAMERA,[39] IPO,[40] MOBCAL,[41] ISiCLE,[42] and DarkChem[43] are command-line tools. Some tools provide both options (e.g., BioTransformer,[44] CFM-ID,[45] etc.), which is the most flexible approach, as it targets all levels of users and can be deployed in a range of different computing environments (e.g., both interactive use on a personal computer and batch processing on a lab server or cluster). A comprehensive survey comparing the strengths and weaknesses of GUIs and CLIs has been previously published,[46] and the Metabolomics Tools Wiki[47] is a useful resource that classifies tools based on their UI and other software features.

In general, GUIs should be implemented in software for basic users in order to guide them through the sequential analysis steps. This makes GUI-driven software easier to learn and simpler to operate.[48] On the other hand, CLI-driven programs are often preferred by software developers and advanced users, as they provide ultimate flexibility to create novel data analysis pipelines and automate tedious procedures through the development of custom scripts that interact with the software programmatically.[49] Furthermore, some GUI or web-based tools are not amenable to high-performance or cloud computing architectures, nor to data management systems that analyze data in real-time as it is produced. Providing multiple interfaces for accessing the same software application can be a successful strategy to meet a broad range of needs as the user base grows. In order to design user-friendly program interfaces, it is recommended to consider the following principles:

Provide a User Interface for Automated Installation. If a software tool needs to be installed on users' computers, an automated interface that guides users step-by-step through the installation process is recommended. As reported by Mangul et al., the number of citations significantly increased when authors provided an easy installation process.[36] The increasing adoption of Docker and cloud technologies will greatly facilitate the installation of bioinformatics software with many dependencies.

Use Intuitive Function Names in the User Interface. The menus and labels in the UIs should be intuitive and clear so that users can understand each operation, and so the processing steps can be easily repeated (thus increasing reproducibility). In addition, it is now a common practice to provide user manuals, FAQs, and tutorials to explain each step, key parameters, and common errors (see Stage 3 for further details).

Classify Parameters and Options as "Basic" or "Advanced". To provide different functionalities, a software tool may have several parameters and options. Classifying these settings as "basic" or "advanced" and providing default values are helpful practices that enable users with little background knowledge to easily navigate the program.[50]

Report Progress and Error Messages Promptly. A software tool should provide real-time processing and error reports so users can easily monitor the status of data processing and respond to any errors during the process. Ideally, these should be saved in a log file to facilitate troubleshooting or recreating the process after the analysis session has ended (see Stage 3).

Provide User Interfaces to Allow the Exploration and Validation of Outputs. While most software tools export results in a plain text file format, it is useful to provide graphical interfaces that allow users to interact with results within the tool (see Supporting Information). For example, it is important for a compound identification tool to provide an interface for comparing search spectra against a library match. For a quantification tool, an interface that allows users to interactively examine the extracted ion chromatograms and their isotopic peaks is critical.

Code to Standards to Ensure Accessibility. For application and web-based software, there are existing standards for ensuring a product is usable by people that require assistive technology. This can include people with sight impairments, using screen readers, but also those with the need to increase font sizes. Color vision deficiency (CVD) is much more common than developers might imagine, and it is important to not use color alone for meaning. CVD-optimized colormaps have been developed for rendering plots of scientific data.[51] For example, an indicator should use text and a color or a shape and a color. By using standards such as the W3C's Web Content Accessibility Guidelines (WCAG),[52] one can create applications that are not only accessible but will work on a variety of devices from computer screens to phones and tablets.

Although software tools with interactive GUIs are desirable for many users, it requires significant effort by developers to implement both efficient computational tools and user-friendly interfaces at the same time. Many metabolomics tools were initially developed to run as command-line driven software and only later were adapted to run within a GUI or web interface to keep pace with popular demand. For example, XCMS was originally a collection of R scripts,[27] while INCA was initially a package of MATLAB functions.[5] As these programs evolved to include user-friendly interfaces, they maintained access to the programmatic layer by exposing core software functions to the user (in the case of INCA) or by continuing to offer the command-line version as an alternative software package (in the case of XCMS). Other software such as MetaboAnalyst was initially released as a web application but now provides access to the underlying R code that is generated by each analysis step, which can be run as R commands using the companion software MetaboAnalystR.[53] An alternative solution is to integrate newly developed tools into existing GUI-driven programs. For example, MZmine[14] and Galaxy-M[54] provide

https://dx.doi.org/10.1021/acs.analchem.0c03581
Anal. Chem. 2021, 93, 1912−1923

easy-to-use graphical interfaces that allow developers to embed their tools and avoid the need to create a standalone GUI.

Develop Simple, Efficient, Modular, and Reproducible Data Processing Workflows. At a high level, data analysis software is successful due to its usefulness, its intuitiveness, and its flexibility. Researchers should *want* to use the software, and the software should not just support but also *encourage* modification to adapt to changing needs. To satisfy these criteria, developers must consider the software experience from the unique perspective of their target user, since each group has different expectations and requirements. From a basic user's perspective, the ideal tool is sufficiently powerful to complete the required task but also does so with minimal complexity. On the other hand, it is sometimes important to provide features that address the computing needs of more advanced users. Otherwise, the software will not be able to adapt along with its users as they become more experienced and as their requirements evolve, leading them to abandon the tool in search of other programs that offer greater flexibility.

Regardless of the expertise of the user, software tools for metabolomics applications should have capabilities to automate repetitive tasks and batch process multiple data files from the same experiment. Critical data pre-processing steps should be streamlined to maximize computational efficiency and minimize the need for user interaction. Data post-processing steps involving statistical or functional analysis should follow a modular design with flexibility to expand as new approaches are developed or in response to user feedback.[55,56] Exposed configuration parameters should be strategically limited and explained in detail, along with potential positive and negative outcomes that can result when making adjustments. For a GUI, this is achieved through tooltips and searchable menus, with in-menu links to more verbose documentation. For CLIs, help outputs at various interaction levels (e.g., module, sub-module) should provide minimal but sufficient guidance with the option to access further detail.

Software should be implemented such that it is modular and thus encourages reusable functionality. This is reflected by meaningful module, class, and function/method names and the use of an intuitive, object-oriented layout that mirrors the typical analysis pipeline. Currently, most metabolomics applications involve customized workflows with modular sequences that go from raw data to a given biological output. For instance, LC/MS raw data analysis, statistical analysis, comparisons across experimental groups, identification and enrichment of metabolites, placing metabolites in the biological context of function and pathways, and developing quantitative models can all be chained together into a workflow paradigm (Figure 2). However, there is no monolithic workflow that suits all conditions. Ideally, each computing task is categorized within an ontological framework that enables researchers to custom-build workflows by association of modules in the desired hierarchical order ensuring compatibility of input-output features at each step. This plug-and-play approach requires the development of application programming interfaces (APIs) that facilitate concatenating software to achieve workflows.

To ensure an efficient and modular software design, one should consider (i) the source and format of input data, (ii) each step required to process the data through various possible workflows, and (iii) the final destination and format of output data. Discrete analysis tasks (e.g., noise mitigation, peak detection, alignment, etc.) should be implemented at the module level, with associated functionality exposed therein. Importantly, this paradigm enables useful compartmentalization of core functionality. For example, a user may only want to use the peak detection algorithms from a given tool and can only do so *easily* if that functionality is reasonably separated from the rest of the processing/analysis modules. The core computational routines should also be separated from the user interface code, which enables the routines to be easily integrated into other programs or accessed through alternative user interfaces or APIs. Thus, the software must work seamlessly as a whole, from data input to final output generation, but each sufficiently distinct piece of functionality must also be usable with minimal inputs/assumptions, independent of the greater workflow.

Most analysis pipelines depend on parameter settings or other explicit interaction from the user, and providing features that automate *data provenance* is key to achieving reproducible results. Designing software to automatically record the data analysis steps and parameter settings involved in producing a given output (e.g., by generating a log file) *ensures* the metadata required for reproducibility is stored and is available to the user if needed. This concept can be further extended to abstract out other sources of *necessary* tedium and/or complexity, for example, benchmarking, fault tolerance, checkpointing and resuming tasks, scalability to high-performance computing resources, etc.

STAGE 3: DISTRIBUTION AND MAINTENANCE

Document Software Thoroughly. Software documentation is a general term for any text, figure, or other content that supplements code and supports the usage of the software tool. Many of the most successful and well-established software tools have invested in good documentation with the use of multiple formats, such as a navigable HTML interface with a quick start guide. However, improper or missing documentation may result in additional effort for the developer to answer user questions or result in users ultimately discontinuing the use of the tool, thereby reducing its impact on the research community.[57] Many metabolomics software tools suffer from inadequate documentation because developers are typically not properly trained in software engineering and principles of good documentation.[35]

Several reviews have highlighted best practices for documentation and promoting software usability,[35,57−59] and these practices also apply to software specifically designed for metabolomics data. Software documentation should be designed to reach a wide audience and use a variety of formats.[57] Documentation can be targeted toward software developers (e.g., comments in the code), beginners (e.g., tutorials, videos), or advanced users (e.g., reference manual).[59] We describe the most common formats below, which are detailed in the references[35,57−59] and can be classified into two main groups: code-related and use-related.

Code-Related Documentation. *Source Code Comments.* This is critical for understanding the specific code in the software. There are numerous online references for code documentation and style, which can be tailored to a specific programming language (R, Python, Java, etc.). Many languages also support the use of "docstrings" (Python, Julia, etc.), which allow the programmer to assess comments during program runtime. There is an additional advantage of automatically

https://dx.doi.org/10.1021/acs.analchem.0c03581
Anal. Chem. 2021, 93, 1912−1923

Table 2. Example Metabolomics Software with Available Documentation Formats

	code-related			details and examples			maintenance and communication			
	source code	progress message	error message	readme	manual	tutorial	news	mailing list	FAQ	summary figure
MZmine 2[14]	Y	Y	Y	Y	Y	Y	Y (website)	Y (GitHub to report issues)		Y
XCMS[27]	Y (via R)	Y	Y	Y	Y	Y	Y (website and register w/email)	Y (register w/email)	Y	
MetaboAnalyst[22]	Y (via R)	Y	Y	Y	Y	Y (R vignette)	Y (website)		Y	Y

generating documentation that is synchronized to each software version (e.g., using Sphinx for Python), so that documentation does not become obsolete over time.

Progress Messages. Software tools should communicate when the software is entering different stages of execution so that users can identify when there is an unexpected event during the program workflow.

Error Messages. Software tools should communicate to the user when problems occur (e.g., data values not supported, algorithmic or numeric problems).

Use-Related Documentation. *Readme File.* This file is typically short and provides basic information about the software (e.g., short description, programming language, versions, licensing, required dependencies) and installation details.

Reference Manual/Users Guide. This is the most comprehensive document that describes the purpose of each function, the expectations of input/output formats, the syntax for running the function, and the explanation of default parameters. Although comprehensive, this document is typically intended for experienced users.

Quick Start/Tutorial/Demo. A software tool should also include documentation for the beginning user. The purpose of a "quick start", "tutorial", or "demo" is to provide example code and data to enable a user to quickly run the software tool on a typical dataset, which should be provided with the software package. The user should be able to run the code and get the same results (e.g., tables, figures, screenshots) that are described in the document.

In summary, we recommend using multiple formats within the categories described above to have the most impactful software. Indeed, many of the most established and commonly used metabolomics software use multiple formats, as well as dissemination strategies that will be addressed in the next section (Table 2). Although much of the documentation burden is on the software developer, it is also critical for the software users to promote reproducible research by providing documentation for how software was applied in publications, reports, and presentations including the (i) tool name and version number, (ii) parameters, (iii) functions called, and (iv) citations for specific functionalities or modules within the software.

Publicize and Disseminate. After the development of new software, it should be widely publicized to maximize use. Below we describe a few approaches for publicizing and disseminating software tools:

Dedicated Website. This will serve as a central reference point for users to access or download the tool. It will also contain news and updates, list version changes, bug fixes, upgrades, or other changes. A regularly updated News section also conveys that the software tool is well maintained. For software suites or software tools with many components, a summary may be useful for users to understand their interoperability (i.e., the context of how the different tools or components work together). This can be in the form of a flowchart that shows the input and output files and formats required for different tools. The website's dashboard view enables one to analyze the behavior of visitors on the site and to identify their location and domain (e.g., university, institute, industry, government). Marketing applications such as "Google Analytics" may also be considered to better track downloads or user traffic. It is recommended to collect detailed information (name, contact information, job function, etc.) from visitors who request to download or license the software using a web form. This is an efficient way to build an email list for sending announcements to potential users or to obtain statistics that are required in reports to sponsors or employers.

Mailing List or User Forum. If the user base is large, an archived mailing list or user forum may be an efficient format to share knowledge among developers, experienced users, and newcomers.

FAQs. A Frequently Asked Questions (FAQ) document or link is another format for communicating answers to common questions from a large user base.

Academic Publications. The traditional approach to publicizing software developed as a result of non-commercial research is to publish a peer-reviewed article that describes each newly introduced tool, typically with examples of its application to a relevant research topic. However, some tools may not be suitable for peer-reviewed publications, either because they represent an incremental improvement to an existing tool or because the developers simply do not have the resources to pursue publication. Under the European OpenAIRE program,[60] the open-access repository Zenodo[61] was developed in 2015 that provides a digital object identifier (DOI) for research software, reports, and datasets. A DOI can be cited by authors to acknowledge their use of a given tool, thus providing a mechanism for tool developers to receive recognition for their work in publications.

Notebooks. Notebooks, such as those created with Jupyter or R Studio, can help demonstrate how to use your software. This can be particularly important for command-line tools. Notebooks can be linked from the dedicated website and hosted on GitHub, MyBinder, or your institution's JupyterHub.

Workshops and Conferences. Another way to disseminate information and obtain feedback about a given tool is to introduce the software in training programs or workshops organized by institutes or research centers.[62] This will provide opportunities for different users to ask questions or provide feedback. Software examples and best practices can also be presented at conferences and meetings.

Community Websites and Social Media. Links to software tools can be placed on scientific community websites, such as Metabolomics Workbench.[32] Video tutorials can be produced

https://dx.doi.org/10.1021/acs.analchem.0c03581
Anal. Chem. 2021, 93, 1912−1923

to highlight specific features of the tool, which can be posted to websites or social media.

Software Maintenance and Sustainability. Bioinformatics software should be usable, accessible, and archivally stable. However, many bioinformatics tools, web servers, and databases become obsolete with each successive year. According to a recent study,[63] citations are the strongest predictor of the availability of a bioinformatics software over time following publication. This directly speaks to the usefulness of a tool—if it is heavily used by the community, it is more likely to be sustained. Therefore, the scientific content and quality of a piece of software play the most important roles in determining the lifespan of the tool. Here we provide some practical tips to help ease the task of long-term sustainability and maintenance of software:

Write Maintainable Code. The initial excitement upon release of a new software tool gradually settles, and developers soon notice that they spend increasing time in the less "glamorous" maintenance cycle of the software such as fixing bugs, addressing bottlenecks, and adding new features. These activities directly define the lifespan of the software. Even though funding agencies have made efforts in improving or hardening pre-existing software, since these activities often do not result in publications, the motivation is not as strong. To minimize time and effort spent in maintenance, it is critical to use an established programming language with a long life span, follow a consistent coding standard, use modular design, separate code and data, write readable and clean code, keep the code simple, avoid deep nesting, and routinely refactor code.

Use Your Tool Regularly. We recommend to routinely use your tools and experience them from a user's perspective, so that small glitches are quickly identified and fixed, and new features are implemented in a timely manner. This is the case for XCMS Online,[21] ISiCLE,[42] INCA,[5] and MetaboAnalyst[22]—the research teams responsible for these tools also use them on a regular basis so that developers and users share the same motivation to maximize the quality and productivity of the software. Many successful bioinformatics tools are created by top researchers in the field to fulfill an analysis need for their group and are subsequently shared with the scientific community.[64] In this case, long-term maintenance is naturally tied to the major research activities of the lab.

Use Version Control and Online Repositories. Project management and version control, which are standard practices in industrial software engineering, are often ignored by small academic groups that are the main contributors of scientific software. This can create barriers to future collaboration and irreproducible research. It is of critical importance to use a well-adopted IDE, such as NetBeans for Java developers, RStudio for R developers, and Atom or PyCharm for Python. The built-in version control systems such as Git[65] can be configured to connect with an online repository such as Bitbucket[66] or GitHub,[67] which are both free for academics. These repositories essentially serve as an online electronic lab notebook for software development.

Embrace Open-Source. The field of bioinformatics has benefited tremendously from free, open-source software repositories: from Perl/BioPerl (for sequence analysis) to R/Bioconductor (for statistics and visualization). Bitbucket or GitHub are also used widely for code sharing. These repositories encourage, and sometimes enforce, standard practices to improve the quality and maintainability of the code. The open-source nature also allows the original software tools to be extended and improved by other groups, such as the case for XCMS[68] and MZmine 2.[69] Open-source licenses are increasingly encouraged or even required by funding agencies and publishers, and some universities have adopted a preferred open-source license for academic developers (MIT, BSD, Apache, etc.).

Plan for Long-Term Maintenance. Many bioinformatics software tools are developed by one or a handful of people, consisting mainly of graduate students or postdoctoral fellows. This can often cause issues when the main developer(s) relocates (i.e., due to graduation or because funding for the project ends). Therefore, it is critical to create a maintenance plan, such as choosing a programming language and framework that are familiar within the team at the development stage, strategically training next-generation developers during the transition period, or transferring the program into an open-source community project when the software grows beyond the maintenance capacity of the original development team (e.g., Cytoscape[70] or Galaxy[71]). To address the needs and concerns related to software development and maintenance, the US National Science Foundation (NSF) has recently funded the conceptualization of a U.S. Research Software Sustainability Institute (URSSI) to serve as a community hub and support scientists to create improved, more sustainable software.[72]

CONCLUDING REMARKS AND FUTURE PERSPECTIVES

We have outlined a series of practical guidelines for metabolomics software development based on the collective experience of the authors. Even though not all guidelines may apply to a given project, we encourage developers to consider how these recommendations can be implemented in order to avoid common pitfalls and avoid wasted time and effort. We have summarized several of the most important guidelines in the Best Practices section that follows. We also encourage advanced researchers to delve into other literature on good programming practices for scientific software development.[35,50,57,58,62] An emerging facet of modern metabolomics software is to provide capabilities for integration with other types of omics data, such as transcriptomics, proteomics, and (meta)genomics. Multi-omics profiling can increase the statistical power of pathway-based analyses, identify co-regulated species that span multiple functional levels and overlay these disparate measurements onto a common network of biological interactions.[73,74] As metabolomics technologies advance from analysis of single experiments to mining information from large, multi-dimensional datasets, there is a growing need for researchers who are trained to merge the knowledge of computer algorithms (e.g., machine learning, kinetic modeling) and software development with an understanding of basic biological and chemical science. In this regard, implementing and sustaining projects that assist with the development of better data analysis tools will become just as important as efforts to promote the collection of new metabolomics datasets.

BEST PRACTICES FOR METABOLOMICS TOOL DEVELOPMENT

- Identify who will use the tool and why, to avoid redundancy.

https://dx.doi.org/10.1021/acs.analchem.0c03581
Anal. Chem. 2021, 93, 1912−1923

- Consider available libraries, long-term maintenance, interoperability, and cloud computing needs when choosing a programming language/environment.
- Use standard file formats for data input and output.
- Provide user interfaces that are intuitive, interactive, accessible, and appropriate for the target audience.
- Plan for simple, efficient, modular, and reproducible data processing workflows.
- Provide advanced capabilities for power users that extend the reach of the software.
- Include capabilities for data export and visualization to promote easy sharing of results.
- Document software thoroughly using a variety of formats.
- Publicize and disseminate the software through websites, publications, notebooks, and workshops.
- Be a user of your own tool, and have a plan for its sustainability and maintenance.

■ ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available free of charge at https://pubs.acs.org/doi/10.1021/acs.analchem.0c03581.

A detailed guide to common data formats used in metabolomics, including a description of vendor and open formats for raw MS data as well as data formats for specific applications such as library matching, functional analysis, property prediction, and data export and visualization, along with program-specific data formats and considerations related to file size and file compression (PDF)

■ AUTHOR INFORMATION

Corresponding Author

Jamey D. Young − *Department of Chemical and Biomolecular Engineering, Vanderbilt University, Nashville, Tennessee 37235, United States; Department of Molecular Physiology and Biophysics, Vanderbilt University, Nashville, Tennessee 37235, United States;* orcid.org/0000-0002-0871-1494; Email: j.d.young@vanderbilt.edu

Authors

Hui-Yin Chang − *Department of Pathology, University of Michigan, Ann Arbor, Michigan 48109, United States; Department of Biomedical Sciences and Engineering, National Central University, Taoyuan City 320, Taiwan;* orcid.org/0000-0003-1767-1874

Sean M. Colby − *Biological Sciences Division, Pacific Northwest National Laboratory, Richland, Washington 99352, United States*

Xiuxia Du − *Department of Bioinformatics & Genomics, University of North Carolina at Charlotte, Charlotte, North Carolina 28223, United States;* orcid.org/0000-0002-3468-9585

Javier D. Gomez − *Department of Chemical and Biomolecular Engineering, Vanderbilt University, Nashville, Tennessee 37235, United States*

Maximilian J. Helf − *Boyce Thompson Institute and Department of Chemistry and Chemical Biology, Cornell University, Ithaca, New York 14853, United States*

Katerina Kechris − *Department of Biostatistics and Informatics, University of Colorado Anschutz Medical Campus, Aurora, Colorado 80045, United States*

Christine R. Kirkpatrick − *San Diego Supercomputer Center, University of California San Diego, La Jolla, California 92093, United States*

Shuzhao Li − *The Jackson Laboratory for Genomic Medicine, Farmington, Connecticut 06032, United States*

Gary J. Patti − *Department of Chemistry, Department of Medicine, and Siteman Cancer Center, Washington University in St. Louis, St. Louis, Missouri 63130, United States;* orcid.org/0000-0002-3748-6193

Ryan S. Renslow − *Biological Sciences Division, Pacific Northwest National Laboratory, Richland, Washington 99352, United States; Gene and Linda Voiland School of Chemical Engineering and Bioengineering, Washington State University, Pullman, Washington 99164, United States;* orcid.org/0000-0002-3969-5570

Shankar Subramaniam − *San Diego Supercomputer Center, University of California San Diego, La Jolla, California 92093, United States; Department of Bioengineering, Department of Computer Science and Engineering, Department of Cellular and Molecular Medicine, and Department of Chemistry and Biochemistry, University of California San Diego, La Jolla, California 92093, United States*

Mukesh Verma − *Epidemiology and Genomics Research Program, National Cancer Institute, National Institutes of Health, Rockville, Maryland 20850, United States*

Jianguo Xia − *Faculty of Agricultural and Environmental Sciences, McGill University, Ste. Anne de Bellevue, Quebec H9X 3 V9, Canada*

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.analchem.0c03581

Notes

The authors declare the following competing financial interest(s): J.D.Y. is a co-founder, shareholder, and director of Metalytics, Inc.

All authors (listed alphabetically) are members of the NIH Common Fund Metabolomics Consortium Software Standards Working Group.

■ ACKNOWLEDGMENTS

This work was supported by NIH grants U2C ES030164 (H.-Y.C.), U2C ES030170 (S.M.C., R.S.R.), U01 CA235507 (X.D.), U01 CA235488 (K.K.), U2C DK119886 (C.R.K., SS), U01 CA235493 (S.L., J.X.), U01 CA235482 (G.J.P.), and U01 CA235508 (J.D.G., J.D.Y.). M.J.H. was supported by a Research Fellowship from the Deutsche Forschungsgemeinschaft (DFG), Project Number 386228702. The authors thank the Metabolomics Consortium Steering Committee for useful discussions and feedback.

■ REFERENCES

(1) Want, E.; Masson, P. *Methods Mol. Biol.* **2011**, *708*, 277.
(2) Johnson, C. H.; Ivanisevic, J.; Siuzdak, G. *Nat. Rev. Mol. Cell Biol.* **2016**, *17* (7), 451−459.
(3) Sugimoto, M.; Hirayama, A.; Robert, M.; Abe, S.; Soga, T.; Tomita, M. *Electrophoresis* **2010**, *31*, 2311.
(4) Creek, D. J.; Jankevics, A.; Breitling, R.; Watson, D. G.; Barrett, M. P.; Burgess, K. E. V. *Anal. Chem.* **2011**, *83*, 8703.
(5) Young, J. D. *Bioinformatics* **2014**, *30* (9), 1333−1335.

(6) Hoops, S.; Sahle, S.; Gauges, R.; Lee, C.; Pahle, J.; Simus, N.; Singhal, M.; Xu, L.; Mendes, P.; Kummer, U. *Bioinformatics* **2006**, *22* (24), 3067−3074.
(7) Heirendt, L.; Arreckx, S.; Pfau, T.; Mendoza, S. N.; Richelle, A.; Heinken, A.; Haraldsdóttir, H. S.; Wachowiak, J.; Keating, S. M.; Vlasov, V.; et al. *Nat. Protoc.* **2019**, *14* (3), 639−702.
(8) Stanstrup, J.; Broeckling, C. D.; Helmus, R.; Hoffmann, N.; Mathé, E.; Naake, T.; Nicolotti, L.; Peters, K.; Rainer, J.; Salek, R. M. *Metabolites* **2019**, *9*, 200.
(9) O'Shea, K.; Misra, B. B. *Metabolomics* **2020**, DOI: 10.1007/s11306-020-01657-3.
(10) Spicer, R.; Salek, R. M.; Moreno, P.; Cañueto, D.; Steinbeck, C. *Metabolomics* **2017**, *13* (9), 106.
(11) Stack Overflow. Developer Survey 2019. https://insights.stackoverflow.com/survey/2019.
(12) Shannon, P.; Markiel, A.; Ozier, O.; Baliga, N. S.; Wang, J. T.; Ramage, D.; Amin, N.; Schwikowski, B.; Ideker, T. *Genome Res.* **2003**, *13*, 2498.
(13) Dührkop, K.; Fleischauer, M.; Ludwig, M.; Aksenov, A. A.; Melnik, A. V.; Meusel, M.; Dorrestein, P. C.; Rousu, J.; Böcker, S. *Nat. Methods* **2019**, *16*, 299.
(14) Pluskal, T.; Castillo, S.; Villar-Briones, A.; Orešič, M. *BMC Bioinf.* **2010**, *11*, 395.
(15) Pittard, W. S.; Li, S. The Essential Toolbox of Data Science: Python, R, Git, and Docker. In *Methods in molecular biology*; Li, S., Ed.; Springer US: New York, 2020; Vol. *2104*, pp 265−311. DOI: 10.1007/978-1-0716-0239-3_15
(16) Smith, C. A.; Want, E. J.; O'Maille, G.; Abagyan, R.; Siuzdak, G. *Anal. Chem.* **2006**, *78*, 779.
(17) CoreMS, https://pypi.org/project/CoreMS/.
(18) Software Carpentry, https://software-carpentry.org/.
(19) Git, https://git-scm.com/.
(20) Vlasov, Y.; Khrystenko, N.; Uzun, D. Analysis of Modern Continuous Integration/Deployment Workflows Based on Virtualization Tools and Containerization Techniques. In *Integrated Computer Technologies in Mechanical Engineering*; Nechyporuk, M., Pavlikov, V., Kritskiy, D., Eds.; Springer International Publishing: Cham, 2020; pp 538−549.
(21) Tautenhahn, R.; Patti, G. J.; Rinehart, D.; Siuzdak, G. *Anal. Chem.* **2012**, *84*, 5035.
(22) Chong, J.; Soufan, O.; Li, C.; Caraus, I.; Li, S.; Bourque, G.; Wishart, D. S.; Xia, J. *Nucleic Acids Res.* **2018**, *46*, W486.
(23) Aprà, E.; Bylaska, E. J.; de Jong, W. A.; Govind, N.; Kowalski, K.; Straatsma, T. P.; Valiev, M.; van Dam, H. J. J.; Alexeev, Y.; Anchell, J.; et al. *J. Chem. Phys.* **2020**, *152* (18), 184102.
(24) Peters, K., et al. *Gigascience*. 2019, https://doi.org/10.1093/gigascience/giy149.
(25) Jupyter, https://jupyter.org/.
(26) Tsugawa, H.; Cajka, T.; Kind, T.; Ma, Y.; Higgins, B.; Ikeda, K.; Kanazawa, M.; Vandergheynst, J.; Fiehn, O.; Arita, M. *Nat. Methods* **2015**, *12* (6), 523−526.
(27) Smith, C. A.; Want, E. J.; O'Maille, G.; Abagyan, R.; Siuzdak, G. *Anal. Chem.* **2006**, *78*, 779.
(28) Wang, M.; Carver, J. J.; Phelan, V. V.; Sanchez, L. M.; Garg, N.; Peng, Y.; Nguyen, D. D.; Watrous, J.; Kapono, C. A.; Luzzatto-Knaan, T.; et al. *Nat. Biotechnol.* **2016**, *34*, 828−837.
(29) Li, S.; Park, Y.; Duraisingham, S.; Strobel, F. H.; Khan, N.; Soltow, Q. A.; Jones, D. P.; Pulendran, B. *PLoS Comput. Biol.* **2013**, *9*, e1003123.
(30) NIST. NIST Libraries of Peptide Tandem Mass Spectra. DOI: 10.18434/T4ZK5S.
(31) MassIVE, https://massive.ucsd.edu/ProteoSAFe/static/massive.jsp.
(32) Metabolomics Workbench, https://www.metabolomicsworkbench.org/.
(33) MetaboLights, https://www.ebi.ac.uk/metabolights/.
(34) Yao, C. H.; Wang, L.; Stancliffe, E.; Sindelar, M.; Cho, K.; Yin, W.; Wang, Y.; Patti, G. J. *Anal. Chem.* **2020**, *92* (2), 1856.
(35) List, M.; Ebert, P.; Albrecht, F. *PLoS Comput. Biol.* **2017**, *13*, e1005265.
(36) Mangul, S.; Martin, L. S.; Eskin, E.; Blekhman, R. *Genome Biol.* **2019**, *20*, 47.
(37) Lai, Z.; Tsugawa, H.; Wohlgemuth, G.; Mehta, S.; Mueller, M.; Zheng, Y.; Ogiwara, A.; Meissen, J.; Showalter, M.; Takeuchi, K. *Nat. Methods* **2018**, *15*, 53.
(38) Chang, H. Y.; Chen, C. T.; Lih, T. M.; Lynn, K. S.; Juo, C. G.; Hsu, W. L.; Sung, T. Y. *PLoS One* **2016**, *11*, e0146112.
(39) Kuhl, C.; Tautenhahn, R.; Böttcher, C.; Larson, T. R.; Neumann, S. *Anal. Chem.* **2012**, *84*, 283.
(40) Libiseller, G.; Dvorzak, M.; Kleb, U.; Gander, E.; Eisenberg, T.; Madeo, F.; Neumann, S.; Trausinger, G.; Sinner, F.; Pieber, T. *BMC Bioinf.* **2015**, *16*, 118.
(41) Shvartsburg, A. A.; Jarrold, M. F. *Chem. Phys. Lett.* **1996**, *261*, 86.
(42) Colby, S. M.; Thomas, D. G.; Nunez, J. R.; Baxter, D. J.; Glaesemann, K. R.; Brown, J. M.; Pirrung, M. A.; Govind, N.; Teeguarden, J. G.; Metz, T. O. *Anal. Chem.* **2019**, *91*, 4346.
(43) Colby, S. M.; Nuñez, J. R.; Hodas, N. O.; Corley, C. D.; Renslow, R. R. *Anal. Chem.* **2020**, *92*, 1720.
(44) Djoumbou-Feunang, Y.; Fiamoncini, J.; Gil-de-la-Fuente, A.; Greiner, R.; Manach, C.; Wishart, D. S. *J. Cheminf.* **2019**, *11* (1), 2.
(45) Djoumbou-Feunang, Y.; Pon, A.; Karu, N.; Zheng, J.; Li, C.; Arndt, D.; Gautam, M.; Allen, F.; Wishart, D. S. *Metabolites* **2019**, *9* (4), 72.
(46) Takayama, L.; Kandogan, E. Trust as an Underlying Factor of System Administrator Interface Choice. In *CHI '06, Conference on Human Factors in Computing Systems*; Association for Computing Machinery: New York, 2006. DOI: 10.1145/1125451.1125708
(47) Metabolomics Tools Wiki, https://raspicer.github.io/MetabolomicsTools/.
(48) Mangul, S.; Mosqueiro, T.; Abdill, R. J.; Duong, D.; Mitchell, K.; Sarwal, V.; Hill, B.; Brito, J.; Littman, R. J.; Statz, B.; et al. *PLoS Biol.* **2019**, *17*, e3000333.
(49) Remington, R. W.; Yuen, H. W. H.; Pashler, H. *J. Exp. Psychol. Appl.* **2016**, *22*, 95.
(50) Ramakrishnan, L.; Gunter, D. Ten Principles for Creating Usable Software for Science. *Proceedings of the 13th International Conference on eScience, eScience 2017*, Auckland, New Zealand; IEEE, 2017. DOI: 10.1109/eScience.2017.34
(51) Nuñez, J. R.; Anderton, C. R.; Renslow, R. S. *PLoS One* **2018**, *13* (7), No. e0199239.
(52) WebAIM. Web Content Accessibilty Guidelines. https://webaim.org/standards/wcag/.
(53) Chong, J.; Xia, J. *Bioinformatics* **2018**, *34*, 4313.
(54) Davidson, R. L.; Weber, R. J. M.; Liu, H.; Sharma-Oates, A.; Viant, M. R. *GigaScience* **2016**, *5*, s13742-016-0115-8.
(55) Cambiaghi, A.; Ferrario, M.; Masseroli, M. *Briefings Bioinf.* **2016**, *18*, 498−510.
(56) Karaman, I.; Climaco Pinto, R.; Graça, G. Metabolomics Data Preprocessing: From Raw Data to Features for Statistical Analysis. In *Comprehensive Analytical Chemistry*; Jaumot, J., Bedia, C., Tauler, R., Eds.; Elsevier, 2018; Vol. *82*, pp 197−225. DOI: 10.1016/bs.coac.2018.08.003
(57) Karimzadeh, M.; Hoffman, M. M. *Briefings Bioinf.* **2018**, *19*, 693.
(58) Taschuk, M.; Wilson, G. *PLoS Comput. Biol.* **2017**, *13*, e1005412.
(59) Leprevost, F. da V.; Barbosa, V. C.; Francisco, E. L.; Perez-Riverol, Y.; Carvalho, P. C. *Front. Genet.* **2014**, *5*, 199.
(60) Manghi, P.; Manola, N.; Horstmann, W.; Peters, D. *Grey J.* **2010**, *6*, 31−40.
(61) Zenodo, https://zenodo.org/.
(62) Yurkovich, J. T.; Yurkovich, B. J.; Dräger, A.; Palsson, B. O.; King, Z. A. *Cell Systems.* **2017**, *5*, 431.
(63) Wren, J. D.; Georgescu, C.; Giles, C. B.; Hennessey, J. *Nucleic Acids Res.* **2017**, *45*, 3627.

(64) Altschul, S.; Demchak, B.; Durbin, R.; Gentleman, R.; Krzywinski, M.; Li, H.; Nekrutenko, A.; Robinson, J.; Rasband, W.; Taylor, J. *Nat. Biotechnol.* **2013**, *31*, 894.
(65) Ram, K. *Source Code Biol. Med.* **2013**, *8*, 7.
(66) Bitbucket, https://bitbucket.org/.
(67) GitHub, https://github.com/.
(68) Tautenhahn, R.; Bottcher, C.; Neumann, S. *BMC Bioinf.* **2008**, *9*, 504.
(69) Smirnov, A.; Qiu, Y.; Jia, W.; Walker, D. I.; Jones, D. P.; Du, X. *Anal. Chem.* **2019**, *91*, 9069.
(70) Saito, R.; Smoot, M. E.; Ono, K.; Ruscheinski, J.; Wang, P. L.; Lotia, S.; Pico, A. R.; Bader, G. D.; Ideker, T. *Nat. Methods* **2012**, *9*, 1069.
(71) Afgan, E.; Baker, D.; Batut, B.; Van Den Beek, M.; Bouvier, D.; Ech, M.; Chilton, J.; Clements, D.; Coraor, N.; Grüning, B. A.; et al. *Nucleic Acids Res.* **2018**, *46*, W537.
(72) Carver, J. C.; Gesing, S.; Katz, D. S.; Ram, K.; Weber, N. *Comput. Sci. Eng.* **2018**, *20*, 4.
(73) Huang, S.; Chaudhary, K.; Garmire, L. X. *Front. Genet.* **2017**, *8*, 84.
(74) Pinu, F. R.; Beale, D. J.; Paten, A. M.; Kouremenos, K.; Swarup, S.; Schirra, H. J.; Wishart, D. *Metabolites* **2019**, *9*, 76.

https://dx.doi.org/10.1021/acs.analchem.0c03581
Anal. Chem. 2021, 93, 1912−1923