Recommendations for the review of archaeological research software

Timo Homburg, Anne Klammt, Hubert Mara, Clemens Schmid, Sophie Charlotte Schmidt, Florian Thiery & Martina Trognitz

Abstract – Motivated by numerous discussions around the increasing use of research software in the field of archaeology, this article outlines some aspects for its review. The evaluation of software is a complex topic, since its field of application and development context has a considerable influence. In addition, there are many very different use cases, ranging from students wanting a quick solution for an exercise to project developers, who have to integrate a software package into an existing infrastructure for continuous operation. Although this discussion paper is based on equal contributions from archaeology and applied computer science, the focus is on evaluation criteria for software in the field of archaeology. A major goal of this paper is to alert future reviewers to the complexity of software evaluation. A software review should enable a professional archaeological audience to make a quick, critical and fair, to both the product and the developers assessment of the software. Priority recommendations include a description of the context in which the review was written and the requirements for specific use cases. In addition, a short tabular overview should enable a quick assessment of the technical, financial and legal aspects. The need for future adaptations of this guideline was identified at the outset since both software development and its evaluation in the digital age are expected to remain very dynamic.

Key words – archaeology; manual; guideline; software review; research software; review

Titel – Diskussionsbeitrag: Handreichung zur Rezension von Forschungssoftware in der Archäologie und den Altertumswissenschaften

Zusammenfassung – Motiviert durch zahlreiche Diskussionen rund um den zunehmenden Einsatz von Forschungssoftware im Bereich der Archäologie werden in diesem Beitrag Aspekte zu deren Rezension skizziert. Die Bewertung von Software ist ein komplexes Thema, da deren Einsatzgebiet und ihr Entwicklungskontext einen erheblichen Einfluss auf Forschungsergebnisse haben. Hinzu kommen unterschiedlichste Anwendungsfälle von z. B. Studierenden, die rasch eine Übungsaufgabe lösen, bis hin zu Projektentwicklern, die ein Softwarepaket für den Dauerbetrieb in eine bestehende Infrastruktur integrieren müssen. Obwohl an der Erstellung dieser ersten Version eines Leitfadens paritätisch Beiträge aus der Archäologie und der angewandten Informatik eingeflossen sind, liegt der Fokus auf Beurteilungskriterien von Software im Anwendungsbereich der Archäologien. Ein Ziel dieses Impulses für Softwarerezensionen ist es, künftige Rezensentinnen und Rezensenten für die Komplexität der Beurteilung von Software zu sensibilisieren. Eine Softwarerezension soll dem archäologischen Fachpublikum eine rasche, kritische und – auch den Entwicklerinnen und Entwicklern gegenüber – faire Beurteilung von Software ermöglichen. Zu den vorrangigen Empfehlungen gehört die Beschreibung des Kontextes, in dem die Rezension verfasst wurde bzw. die gestellten Anforderungen für bestimmte Anwendungsfälle. Zusätzlich soll eine kurze tabellarische Übersicht eine rasche Einschätzung der technischen, finanziellen und rechtlichen Aspekte ermöglichen. Der Bedarf nach künftigen Überarbeitungen dieses Impulses für Softwarerezensionen wurde bereits bei dessen Erarbeitung festgestellt, da sowohl bei der Softwareentwicklung wie auch deren Bewertung im digitalen Zeitalter weiterhin eine große Dynamik zu erwarten ist.

Schlagwörter – Archäologie; Archäoinformatik; Handreichung; Softwarebewertung; Forschungssoftware; Rezension

Objectives of this contribution

The aim of this paper is to stimulate discussion about the content and objectives of research software reviews as well as digital tools used for research in archaeology and ancient studies. Our reflections were inspired by a round-table discussion led by Kai-Christian Bruhn, Sophie Charlotte Schmidt and Frank Siegmund at the plenary session of the 9th Workshop of the German Section of the CAA 2019 in Wilhelmshaven. On the occasion of the introduction of the new category "Archäoinformatik" as a space for such kind of reviews in the *Archäologische Informationen* , we have now written down our thoughts. The few relevant software reviews available at the time of our discussion (especially in the journal Internet Archaeology) do not seem to be consistent and comprehensive enough to the authors.

In this recommendation, we present criteria and principles that make up good research software and may be relevant for its evaluation. We also recommend a procedure for an actual software review, and present a list of questions to help with the critical evaluation of software.

The assessment of a publication is a traditional part of academic discourse. Accordingly, there is an unspoken understanding of what a review should include. In addition, concrete measures can be taken to ensure quality, such as instructions on how to write reviews or - as practised by the editorial staff of *Archäologischen Informationen* - the peer reviews are themselves assessed.

What constitutes a good review of software intended for use in archaeological research? Should only the algorithms executed by the software which directly contribute to the solution of a scientific problem be addressed? Or do aspects of

Received: 23 Okt 2020
accepted: 10 Nov 2020
published online: 19 Jan 2021

usability, sustainability and interoperability also need to be considered? What role do technical and legal aspects play in the discussion, for example, source code documentation or licensing? And finally: does the software itself perhaps represent a scientific contribution? Which achievements of the software developers should be considered in a review? What standards, alluded to in the recommendations of the German Research Foundation (Deutschen Forschungsgemeinschaft, DFG), should good research software meet? And last but not least: who should, or is able to, adequately review a piece of software?

The central question of a review, namely the scientific value of the subject to be reviewed, is only touched upon with these questions. In this respect, the requirements of a software review are the same as those of a scholarly text review. Furthermore, due to the variety and diversity of software, not all of the criteria collected here will be relevant in each and every review. It is up to the reviewer to select relevant aspects and to set priorities.

This contribution is intended as a stimulus to discussion. Contributions for discussion and suggestions are explicitly welcome. For this purpose, the authors can be contacted via the given addresses or by posting comments directly on GitHub (https://github.com/Research-Squirrel-Engineers/Impuls_SoftwareRezensionen_DGUF/ [23.10.2020]).

Research software

The criteria for the assessment of software presented below are primarily aimed at reviewing research software. By research software, we mean software that has been developed with research activities as a primary area of application, i.e. generating, processing or analysing research data (Hettrick et al., 2014). Examples include programmes that serve to calibrate and convert measured values, visualise geospatial data, annotate texts and objects or provide and link subject-relevant vocabularies.

Research software is always part of the research process, and has to be comprehensible and - as far as possible - reproducible in all its aspects.

Research software is different from software that is required for the use of specific hardware equipment, such as for surveying, photographic documentation or the analysis of surfaces and substance components. The latter is often proprietary and is distributed together with the hardware. Although it is used in the research process, it does not correspond to our understanding of research software. Digital tools that play a significant role in practical work but have no actual part in the collection, processing and analysis of data are also not research software as we understand it. These include, for example, word processing or spreadsheet programmes. However, the criteria presented can be used to assess any type of software.

Research software is particularly prone to issues of sustainable development and maintenance. One major reason is the lack of long-term funding for staff and infrastructure to produce sustainable software (Anzt et al., 2020, p. 2). An overview of needs, challenges and possible solutions for sustainable research software development was compiled in 2019 in a workshop at the deRSE conference in Potsdam (https://de-rse.org/de/conf2019/index.html [23.10.2020]) (Bach et al., 2019).

Research software as a scholarly achievement

In modern research, it is unthinkable to work without digital tools. This also applies to historical, classical and ancient studies. With the rise of the digital humanities, software is increasingly becoming an important part of the research process and substantially affects it, both implicitly and explicitly (for archaeology, see e.g. (Schmidt and Marwick, 2020). Only by disclosing the underlying source code can the processes performed by the software be properly evaluated.

Despite the vital role that research software plays in many projects, the achievements of those developing and programming it are often not sufficiently recognised academically (Hettrick, 2016; Katerbow and Feulner, 2018; Scheliga et al., 2017). This does not do justice to the fact that scientific expertise and, moreover, advanced technical competence are required to develop research software. Practice and knowledge are explicitly manifested and developed by turning it into code. We argue that these accomplishments need to be acknowledged and made visible, especially as scientific breakthroughs are often only made possible by software (Helmholtz Open Science, 2019; Katerbow and Feulner, 2018; Scheliga et al., 2017).

As an important first step towards making such achievements visible, the job title of *"Research Software Engineer"* (RSE, RSEng) was coined in 2012 (Baxter et al., 2012; Hettrick, 2016). At the same time, an active, interdisciplinary community has formed to develop recommendations for dealing with research software (Anzt et al., 2020). National RSE sections (https://sorse.github.io/

contact/chapters/ [23.10.2020]) – in Germany for example "*de-RSE e.V.* " – organise cross-disciplinary RSE conferences such as SORSE (https://sorse.github.io [23.10.2020]). As part of the development of a Nationale Forschungsdateninfrastruktur (NFDI, Eng. National Research Data Infrastructure), research software and RSEs play a major role in the further development of the research landscape and the portfolios of research institutions in Germany (Goedicke and Lucke, 2020; Löffler, 2020).

One of the demands of the de-RSE e.V. is that research software shall become explicitly visible by means of suitable publication modalities. A publication increases discoverability and thus avoids redundant redevelopments (Anzt et al., 2020, p. 10). At the same time, the clear authorship that goes hand in hand with a publication is of great importance in linking the task of developing software with the responsible people. This is a critical part of establishing an academic career, but up to now only classic text publications have been counted. In future, data publications, software development (Druskat et al., 2017) and annotations — as well as their citations (https://citation-file-format.github.io [17.12.2020]) — should be criteria for the assessment of scientific performance (NFDI4Culture, https://www.rse4nfde.de/en [23.10.2020], RSE4NFDI, (https://www.rse4nfdi.de/de/index.html [23.10.2020], NFDI4Objects, https://www.nfdi4objects.net [23.10.2020]). Some technical challenges have to be overcome though, because in contrast to most text publications software is never finally completed, often builds upon existing modules and is sometimes maintained by changing teams over decades (Katz et al., 2016).

Openly available software can be reviewed, just as is the established practice in archaeology and ancient studies research. It can then be presented to a wider audience in a review. Through a dedicated review of software analogous to reviews of scholarly publications, the scientific achievements, responsibilities and participation of the authors of research software are made visible and recognised by means of familiar formats.

Good research software

In order to be able to establish criteria for reviewing and evaluating research software, the first question is how to determine its quality and by which standards it should be assessed. This includes a multitude of specialist and technical aspects, which will be examined below. However, software, like any other tool, must first be evaluated in terms of its general suitability for sustainable and ethical scientific work.

With its "*Leitlinien zur Sicherung guter wissenschaftlicher Praxis*" (Eng. *Guidelines for Safeguarding Good Scientific Practice*) (DFG, 2019), the DFG formulated a set of guidelines for good scientific research practice. The guidelines correspond to an international consensus and well established principles. The requirements include, for example, adherence to and definition of standards and methods, comprehensible documentation of how results were achieved, public accessibility of results and archiving of the necessary materials to replicate the results.

Some of the criteria that are explicitly and implicitly mentioned in the DFG guidelines are described in more detail below, including the FAIR Data Principles mentioned in the comment for guideline 13, and the principles of Open Science and the CARE principles in guidelines 2 and 10. Further guidance on the implementation of the DFG guidelines in relation to research software can be found at (forschungsdaten.info, 2020).

FAIR Data Principles

The *FAIR Data Principles* (Wilkinson et al., 2016) were published in 2016 and primarily target research data and their metadata. The principles require that data and metadata must be findable, accessible, interoperable and reusable. There is a particular focus on machine readability. The principles can also be applied to research software and its metadata (Goedicke and Lucke, 2020; Lamprecht et al., 2019) and thus provide criteria for assessing the quality of software publication, code documentation and programming standards.

Software becomes findable by storing it, together with its associated metadata, in dedicated software repositories (e.g. GitHub). Integration into scientific research infrastructures increases the findability, as they explicitly work on the development, maintenance and application of controlled indices and metadata standards (Anzt et al., 2020; Lamprecht et al., 2019).

The citation of research software is enabled through persistent identifiers (e.g. URIs) assigned by repositories and crediting the authors. Since software, unlike conventional publications, is constantly being developed and evolves, it should be saved in different versions that can be individually cited (forschungsdaten.info, 2020).

Public repositories and persistent identifiers also ensure a minimum level of accessibility. Ideally, data and metadata should also be machine-accessible via an open and standardised communication protocol (Lamprecht et al., 2019). Other aspects that facilitate accessibility are availability for different operating systems and technical requirements that can be met by current devices in use. Furthermore, a comprehensible user manual improves accessibility.

Interoperability of software can refer to the compatibility of input and output formats with other programmes in a work process (horizontal dimension), but also to the cooperation of the components used in the software itself (vertical dimension) (Lamprecht et al., 2019, p. 46f). The use of standards enables both, as it allows software components to work together even across different operating systems.

The reusability of software depends on several components. Metadata and comprehensive documentation of the software should enable others to reproduce results, process their own data and use it for modified use cases. Additionally, a suitable licence that also takes into account the software dependencies provides information about which rules apply to the use and further development of the code (Lamprecht et al., 2019, pp. 48–49).

Open Science

Open Science refers to a scientific practice that aims at a transparent, reproducible and collaborative research process (Bezjak et al., 2018, 'Open Concepts and Principles'). Open Science consists of several principles that concern different stages in the research process. For example, Open Science calls not only for the results to be openly available (Open Access), but also for opening the underlying data (Open Data), the methods employed (Open Methodology) and the research software used (Open Source). For research software to be open, its source code must at least be accessible and provided with a licence that allows further development (Bezjak et al., 2018, 'Open Research Software and Open Source').

The demand for the disclosure of source code (Open Source) also complies with the idea of reusability. The ideal of Open Source is expanded by the demand for making the code freely available for use (free source) (Stallman, 2001). This is referred to as FOSS (Free/Libre Open Source Software) and is advocated in Germany by, for example, application-oriented scientists and developers from the field of geoinformatics (FOSSGIS e.V., https://www.fossgis.de [23.10.2020]) as well as by research software engineers in the *"de-RSE e.V."* (https://de-rse.org/en/association.html [23.10.2020]; Anzt et al., 2020).

The DFG guidelines touch upon the opening of software source code in the context of quality assurance (DFG, 2019, pp. 14–15) and the creation of public access to research results (DFG, 2019, p. 19).

CARE Principles and ethos

In DFG guidelines 2 and 10, ethical aspects in the research process are addressed. We believe that these should also be taken into account in the area of research software.

Privileges and inequalities in classical and ancient studies research are unconsciously perpetuated in research software as well. Expensive, proprietary software disadvantages researchers who have to make do with less funding, as this denies them access to contemporary analytical methods. As a further consequence, access to highly rated journals may be impeded and student training in these methods may stagnate, further accentuating existing imbalances. Language barriers also contribute to disparities: for example, if international teams use a database that is not also available in the local language, Indigenous researchers may be prevented from independently analysing and evaluating the data, as well as from pursuing further training.

In the field of archaeology, where research is often conducted on foreign cultures and cultural legacies, locals and Indigenous people should be given the opportunity to have a say in or participate in its investigation. To meet this ideal, suitable orientation is provided by the CARE Principles (https://www.gida-global.org/care [23.10.2020]). They were jointly introduced by the Global Indigenous Data Alliance (GIDA) (https://www.gida-global.org [23.10.2020]) and the Research Data Alliance (RDA) (https://www.rd-alliance.org [23.10.2020]) in 2018 on the basis of the UN Declaration on the Rights of Indigenous Peoples (UNDRIP) (https://www.un.org/development/desa/indigenouspeoples/declaration-on-the-rights-of-indigenous-peoples.html [23.10.2020]). Like the FAIR Data Principles, the CARE Principles focus explicitly on research data. However, in our opinion, the four pillars of Collective Benefit, Authority to Control, Responsibility and Ethics are equally relevant to research software.

A possible approach to reviewing archaeological research software

The review of archaeological research software requires sifting through documentation and publications as well as actually testing the software itself. We suggest that a software review, much like a conventional review of a scientific publication, should first introduce the software by presenting the key data. This should be followed by outlining the context and critically assessing the software.

The key data, which can be presented in tabular form, include information on the software version under review, the developers and the licence. This allows, for example, a quick assessment of whether the reviewed software is compatible with one's own technical environment. A suggestion for the contents of such a table can be found at the end of this article.

For context, a classification in the archaeological research field and information on the possible connection with research projects, working groups or institutions should be provided. The background of the reviewer is also necessary; it should be described, because a realistic and transparent assessment of one's own competences and interests regarding the use of the software provide valuable information for a heterogeneous readership. Was the review written purely from a user's point of view or also from a developer's perspective? A description of the test environment used, i.e. the computer used to test the software, can also be important, such as information on the operating system, RAM, processor, graphic card or bandwidth.

A catalogue of questions for the critical assessment of research software from different perspectives, which is divided into three sections follows hereafter. In our opinion, the questions from the first section "Use in archaeology and scientific purpose" should be addressed in every review. We also consider it very important to answer questions on installation, tutorials, and the supporting community, as well as on input and output formats, programming interfaces, and the possibilities for participation in the further development of the software. These questions themselves entail further relevant follow-up questions; for example for the installation process one also has to consider whether the software is a stand-alone or a web application, whether the installation requirements are clearly documented and whether the documentation is complete and up-to-date. The questions in the catalogue we consider the most relevant are prefixed with ✚ (very important) and + (important).

The results of the review should be summarised in a statement that evaluates the software in terms of its usefulness, its usability, its quality of craftsmanship and its position in relation to the aforementioned ideals of good research software.

The catalogue of questions presented below is extensive and does not need to be worked through in every detail. Its complexity and depth is a good indicator that the development of research software can indeed be a full-fledged, scientific activity. We advise critically reflecting on one's own competencies in regard to the review and, if necessary, carrying out the evaluation in a team. For, just as reviews of conventional publications are ideally written by experts on the topic, this should also apply to research software reviews.

Catalogue of questions for the assessment of software

Below we present an annotated catalogue of questions with criteria for the assessment of archaeological research software. Three sections bundle questions from different areas of competence. The first two sections deal with the scientific field of application, as well as the utilisation and usability from the user's point of view. The third section focuses on questions that are particularly relevant for developers and IT administrators. The catalogue concludes with a list of key data that supplement the review in tabular form, but cannot usually be critically assessed.

As with the review of a scientific publication, the composition and weighting of the individual features is to be determined during the review itself and to be set in relation to the context of the application area. Accordingly, we intend the annotated catalogue to be a maximum version that can serve as an aid for the review and the assessment of one's own competences.

Use in archaeology and scientific purpose

When assessing the scientific quality of research software, two central questions must be answered. The first one - and this is a priority for research software - is whether the software contributes to working on archaeological questions in a meaningful way. Secondly, it has to be determined whether the software properly implements the intended task. Both questions are not always easy to answer.

- **What task is the software trying to solve?** This question is linked to the descriptive character of a review: what tasks are being addressed by means of the software in the acquisition, processing or analysis of data? How relevant are the tasks in an archaeological context and how often are they performed? This point illustrates why it is of particular value when archaeologists write software reviews for themselves and their colleagues.
- **How does the software solve a given (technical) task?** A detailed answer to this question is only possible from the developer's perspective. However, core components can usually be easily identified. How is the general software mode of operation designed? What are the basic technical components in the user interface and in the data processing modules behind it? For example, is it a web application that acts as an interface to a database? Or is the software a simple, monolithic command-line programme?
- **How does the scientific workflow implemented in the software work?** This is less about the actual technical implementation than the general methodology employed. Which integral process is applied to data to solve a certain task? Which statistical tools are used? Is there comparative data? An example could be cleaning up input data, subsequently classifying it with an algorithm and finally visualising it.
- **Is the claim to be able to answer a certain scientific question with the chosen workflow correct?** Research software usually promises – implicitly or explicitly – to enable or at least simplify the answering of scientific questions. A GIS application for analysing point patterns, for example, may have been designed for the purpose of demonstrating human settlement behaviour. The question of whether the algorithm used is even capable of doing this may far exceed the scope of a review. Nevertheless, the reviewer should try to make an assessment of plausibility or at least reveal the problem of assessability itself in order to raise awareness for readers. For the assessment, it may help to consider which conclusions could be theoretically deduced from the raw, unprocessed input data in the first place.
- **Have the algorithms been implemented correctly?** The correctness of software results is sometimes difficult to judge because it would require an extensive black-box test. A lack of comparable software with the same functionality makes this even more difficult. Are the algorithms mentioned in the documentation implemented without errors in the source code? Is the scientific result comparable to other software solutions despite all technical differences? Are the algorithms used sufficiently documented and scientifically proven? This question, too, may not be answered conclusively in the context of a review. A prominent indication of a faulty implementation is the lack of robustness, which will be discussed later.
- **Are there any projects/application examples relevant to archaeology in which the reviewed software has already been used?** Software is often developed by and for specific research projects. To assess its quality and relevance, it can therefore be helpful to take a closer look at the research projects themselves. How plausible are the results achieved that are related to the software? How were they received by the scientific community?
- **In what form is the software published?** Ideally, research software should also be scientifically published. This facilitates its citation. Is there a benchmarking paper in which the tool is explicitly presented and compared with alternative products? Are individual versions of the software referenceable separately? For example, does a Digital Object Identifier (DOI) and thus a persistent link exist? Has the software been published in a journal or in another medium? Did it undergo a software peer review mechanism?

Usability and target group orientation

The usability of research software is of central importance because it is a bottleneck that determines the interaction between the user and the software. It includes, for example, the complexity of the installation process, the user interface and machine interfaces. Here it should be noted that, for example, a graphical user interface offers advantages for occasional users, while operation via a command line for processing large data sets is more advantageous for experienced users. Help features (forums, FAQs, tutorials), and the size and activity level of the user and developer community are crucial for practical operability. The amount of activity in the communities can be an important indicator for the sustainability of software and thus its suitability for use at an institutional level in long-term projects.

The following questions concern technical features that influence the user experience (UX). We look at other technical features from the developer's perspective later.

Installation

Installation is often the first step in using the software. Already at this point, issues can become apparent and/or be of importance for the readers of a review.

- ✚ **How does the installation work and where is the software kept?** The easier the installation of the software, the larger its potential user base. From a developer's point of view, the diversity of computer systems often hinders ensuring easy installation for all users. For the review, it can be checked whether an installation script, wizard or package is available. Maybe the software is only available as source code, which must first be compiled by the user; installation scripts or packages allow a much wider user community, whereas compiling allows skilled users to perform the installation on different devices. Specific problems and weaknesses in the installation process can be identified and documented in the review if the software is actually installed by the reviewer.
- \+ **Is it a stand-alone software or a web application?** Not all software needs to be installed locally to be used. Nowadays, web applications that run either dynamically in the browser or on a server are capable of arbitrarily complex operations. Thus, it should be asked whether the application is more suitable for its use in archaeological practice as a local installation or as a web application. Web applications, for example, may not always perform adequately as they require the transfer of large amounts of data over the internet. They may not be executable in the field and are therefore not useful for all work settings.
- \+ **Are necessary requirements in terms of hardware and operating system clearly documented?** Especially during project planning, it is important to be able to correctly identify technical and financial requirements. A specification or instructions on the required hardware and operating system are relevant, for example, for integration with existing infrastructures in larger projects or institutes.

Interface

The usability of software is determined by the possibilities for communication between humans and machines, i.e. the user interface (UI). In some cases, this is supported graphically (GUI), or the operation is carried out via input in the command line interface (CLI). Mixed operation modes are also useful to address different user groups. These are often diverse and can never be clearly defined, for example, excavation technicians or state archaeologists, but for usability analysis they can be used as models to evaluate shared requirements for the software. The design of user interfaces and user navigation in menu structures represents a field of expertise on its own within software development. Solutions that are well adapted to the user's needs are the result of precise knowledge of the target group and their habits and needs. Good solutions, for example, fit seamlessly into the archaeological research process. A good interface supports error-free and efficient work; the comprehensibility of menu items or commands, or the meaningfulness of error messages, are aspects that should be considered in the review. Accessibility for people with disabilities has so far been an insufficiently covered criterion, but it also affects efficiency and possible user groups.

- \+ **Is the user interface suitable for the user group?** Every user group, including the one to which the reviewer belongs, will have certain expectations of the application handling. For some, a command line application is very easy to use, while others will have problems with it. Many software solutions have various user interfaces; for example, many web applications can be accessed both via a search box and filter functions in the graphical interface of a web page as well as code-based via a REST service.
- \+ **Is use in archaeology intended?** The question about the user group should also be asked specifically for archaeology: does the archaeological use correspond to the use scenarios that the developers of the software had in mind? This often has a great influence on the user interface; CAD software frequently used by archaeologists, for example, was designed for architectural or mechanical engineering applications and confronts archaeologists and excavation technicians with an overwhelming plethora of functions.
- – **Does the menu navigation follow certain de-facto standards?** If menu navigation or shortcuts are based on well-known and widespread software in the community, familiarisation with the software is eased and accelerated; some tools deliberately use input masks modelled on widely used spreadsheet programmes.
- – **Is the programme multilingual, and in which languages is it offered?** Depending on the user group, the programme should perhaps be multilingual. Usually, at least an English version is expected. In the case of multilingual software, a usable and readable layout must be created for each language; button labels, for example,

can vary greatly in text length and may result in a truncated display in some languages.

– **Are error messages easily understandable by reviewers?** An error message can enable users to correct the error's cause. Useful and helpful error messages are phrased in an understandable way and visibly placed in the application. Where appropriate, an error message can also provide feedback to the application developers. It must communicate the error message to the user and also send a useful report to the developers to fix the error. Stack traces (references to the position in the programme code where a specific error occurred) or error messages taken from the runtime environment are incomprehensible to most users.

Performanz and robustness

Performance and robustness influence how the programme is used. Performance especially plays a role in the processing of large data sets, while robustness influences the user's saving and back up behaviour.

– **Is the implementation performant?** The importance of the performance of an application depends on the application type. Here, a reviewer should assess whether the software performs its task in reasonable time. If necessary, the execution time of other related software implementations can be compared with the one under test. The reasons for poor software performance are often not easy to identify. For web applications and plugins, a check for responsiveness and good performance across browsers should be executed.

+ **Is the software robust?** The robustness of software essentially requires that the intermediate states of tasks are regularly saved in order to be able to pick up again from the last state in the event of an unforeseen termination. Without this function, an event such as a power failure can cause the loss of settings, data and/or the entire reinitialisation of a calculation. An example of such robustness is the regular automated buffering in a text processing programme. The text document is thus recoverable in the event of a software crash. Depending on the use case, it may also be advantageous for the software to keep a history of user changes and allow users to restore them. These could be language configuration or the settings of units of measurement, specifications for the storage location and the like.

Help features, tutorials and community

In addition to help functions and tutorials, it is of great importance whether the software is supported by a community. The number of active users and developers of a software tool is decisive for whether one can find help in forums when problems arise or, in the case of a small group of users, can only get help through personal exchange. However, small user networks can offer the great advantage that issues are quickly taken up by the developers.

✚ **Are there enough tutorials for learning the software?** Tutorials are essential to address both users and software developers. Users usually expect an easy-to-understand example, focused on the essentials, to get an idea of the typical usage. For developers, it is crucial that a tutorial explains existing programming interfaces (APIs). Good tutorials explain the required knowledge prerequisites and give references to sources for acquiring it. Help in an FAQ or a troubleshooting section also increases the quality of a tutorial. The languages in which the tutorials are available are also of relevance.

+ **Do test data sets exist for the software?** This question is closely related to the question of tutorials, since the latter often work with exercise data. These exercise data or test data sets should be oriented towards scientific practice, but should be understandable without specific prior knowledge. They should be freely available and, if possible, usable without registration.

– **Is further information on the software easy to find?** Do the software information pages or its tutorials refer to further materials? Do they refer to publications by the developers themselves, as well as reviews of them?

✚ **Is the software supported by a community, and what proportion are classical and ancient studies scholars?** Examples of software development that was initially run by a classical and ancient studies community and has since expanded to include other disciplines are Pelagios Commons (https://pelagios.org/ [23.10.2020]) and the web application Recogito (https://recogito.pelagios.org/ [23.10.2020]). Dedicated user groups with a focus on archaeological issues have also developed, for example, within the communities for software packages like QGIS or R.

+ **Are there archaeological best practices or publications that refer to the reviewed software?** While forums, blogs and related media offer direct and often also rapid exchange with users and possibly developers, the inclusion and rec-

ommendation of programmes in best practices and publications is a further indication of their dissemination, scope and reliability.

Data ingest, interoperability and programming interfaces

Input and output data formats affect compatibility with other applications and should be mentioned in a review. In many cases, the supported file formats can be found via a corresponding menu entry (e.g. Save As). The various ways of reading in the data and programming interfaces are also relevant.

+ **Which data formats are read in and how?** Are all relevant data formats supported by the software for the envisaged task and the anticipated typical use case? Are open data formats supported? Can data formats also be read in from common repositories (e.g. web services, Git, cloud services)?
+ **Which output data formats are supported?** The programme should offer output data formats that allow further processing in other (also open source) software packages. Thus, at least one openly specified format should be supported. If data exports are only possible in a proprietary format, maybe even custom to that software, this must be well justified by the developers.
+ **How can data be read in?** Does the software allow batch processing? For many application scenarios, the execution of a once-defined workflow on a batch of files or a series of data is important; such as applying the same transformation steps for all images in a directory.
+ **Is there an application programming interface (API)?** In addition to operability by a human, machine control of the software is also important. This is the only way, for example, to completely automate complex processes with several software components. An API enables this. It should be as open as possible and, if necessary, documented by means of a standard such as OpenAPI (https://www.openapis.org [23.10.2020]). An example of APIs are the endpoints offered by Wikidata, which allow for automated data queries.

Conformity with regulations on data protection and data minimisation

The question of data protection often determines whether the software may be used at all in university research and teaching. However, an assessment of the regulations, unless they clearly refer to the European framework directives, is sometimes very difficult and requires legal advice. The review can already provide an important service for readers by just highlighting the topic. This also applies to data minimisation, necessity of registration processes, cookies and the like.

— **Does the software comply with the laws (e.g. on data protection, map displays, etc.) of the country of assignment?** It must be expected that the software will be used in different regions and countries, each of which has specific laws that affect the execution/installation of the software. An example are cloud applications developed for a North American user base that are not compatible with the EU-wide General Data Protection Regulation.
— **What data does the application store, for what purpose and for how long? Are data transferred to third parties?** In many software applications, usage and user data is collected anonymously to improve the user experience. However, some software providers collect much more data and, for example, also transfer it to third parties. What data is collected and for how long it is stored should become clear from the software documentation. This also applies to the first visit to a web application by a user, at which point the user's consent should be requested. Furthermore, a review could question whether the declared purpose of the data collection is justified and really serves to improve the software.

Developer perspective

Developers are also users of software, but generally have a different perspective on it. This user group has specific technical interests and can often envisage a broader set of application scenarios. A good summary of the developer perspective on software quality is provided by (Jung et al., 2004). There, the specifications of the ISO/IEC 9126 standard (https://en.wikipedia.org/wiki/ISO/IEC_9126 [23.10.2020]) are elaborated with examples.

The following questions focus on the source code and the software architecture. These can only be assessed if the source code is openly accessible (open source), which is usually not at all, or only to a limited extent, the case for proprietary software.

Documentation and tests

Extensive documentation provides a comprehensive understanding of the software's purpose, matureness and current state of development. It is also essential for extending the software. Finally,

solid documentation will also appeal to a larger community of potential developers.

There are several components of documentation, namely documentation of:

the *source code*, i.e. of classes or individual methods,

the *build process*, i.e. how the software is compiled (built) from the source code,

the *software testing process*, i.e. which test cases were considered by the software.

Finally, there is a developer documentation with usage examples.

Software repositories such as GitHub (https://github.com [23.10.2020]) or GitLab (https://gitlab.com [23.10.2020]) often offer templates or best practices to implement these documentation requirements in different programming languages.

An important concept in the context of documentation and testing is that of Continuous Integration (CI). It refers to the continuous assembly of individual application components. For this purpose, routines are prepared to execute various tasks. This allows for automatic triggering of, for example, the creation of source code documentation, the execution of a software test or the generation of an executable file from the source code (release file, EXE file). Usually, the routines are executed again after each change to the source code. By doing so, all programme components are kept up to date.

– **Does a source code documentation exist and, if applicable, is an HTML variant of it available?** Best practices here are, for example, source code documentation with Doxygen (https://www.doxygen.nl [23.10.2020]), JavaDoc (http://www.oracle.com/technetwork/java/javase/documentation/javadoc-137458.html [23.10.2020]), JsDoc (https://jsdoc.app [23.10.2020]) or the popular ReadTheDocs (https://readthedocs.org [23.10.2020]) for Python. All of the tools above generate an HTML representation of the documentation, which should then ideally be made available online, for example as a GitHub page.
– **Is the build process documented and, if applicable, automated by means of build scripts?** In addition to a basic understanding of the programme architecture, knowledge of the build instructions of the software, i.e. the build process, is important. The documentation should include information about a functioning build process of the software. In the past, instructions in README files or similar natural language descriptions were used. Nowadays, it has become an established standard to provide machine-readable build instructions (build scripts). These describe, in an unambiguous, machine-readable way, how the software was built and often allow the build process to be started with a single command. Build scripts also document the dependencies of the software and the libraries used. Examples of such scripts can be found in Apache Maven (https://maven.apache.org [23.10.2020]) or Gradle (https://gradle.org [23.10.2020]).
+ **Is the documentation up to date and does it address all functions of the programme?** The latest edit date for documentation can usually be found in a timestamp if the documentation was created with one of the documentation tools mentioned above. This date should be the same as, or more recent than, the release date of the current software version. If, in the best case, the principles of Continuous Integration are applied, the generation and provision of documentation is directly integrated into the development of the application, such as the GitHub repository of the SPARQLing-Unicorn- QGIS plugin (https://github.com/sparqlunicorn/sparqlunicornGoesGIS [23.10.2020]). Here, an automated process recreates the documentation every time the source code changes and publishes it to the GitHub page of the repository (https://sparqlunicorn.github.io/sparqlunicornGoesGIS/ [23.10.2020]).
– **Is there developer documentation that promotes further software development?** The developer documentation represents the entry point for developers to engage with further enhancement of software beyond just using it. A meaningful README file that briefly demonstrates the use of the programme with the default settings is a minimum requirement. An example of this is Bibtex_JS (https://github.com/pcooksey/bibtex-js [23.10.2020]). Ideally, sample data is included for a better understanding of the programme flow and, if necessary, other frequently used use cases of the software are presented in examples. This applies to the example just mentioned: Bibtex_JS-Examples (https://github.com/pcooksey/bibtex-js/tree/master/test [23.10.2020]). Depending on the complexity of the software, it may be useful to provide a wiki, possibly also maintained by a user community, to explain advanced options (see Bibtex_JS-Wiki; https://github.com/pcooksey/bibtex-js/wiki [23.10.2020]).
– **Does the source code contain software tests for testing core functions and demonstrating them to other developers?** Any software can be checked for its expected behaviour with

software tests. Accordingly, software tests provide information about the envisaged use cases and, as a test result, which functions are stable, error-prone or in need of improvement. As part of a Continuous Integration process, software tests can be executed automatically after each change to the source code.

– **Is it made easy for the developer to test the software (e.g. virtual machine, Docker container, installer)?** Developers are usually able to build and run software, but this can be a more or less complex task. In fact, software that is easy to install and, more importantly, easy to test will be more popular with developers who want to get a quick look at the functionality, as well as with other users. Obviously, available testing methods depend on the type of application. A web application, for example, may be hosted on the Internet and provide the user with a test account (e.g. CWRCWriter; https://cwrc.ca [23.10.2020]), whereas a desktop application may already include an installable application or installer package. Server applications can be available for easy testing as virtual machine images or, more commonly lately, as Docker images in portals such as Dockerhub (https://hub.docker.com [23.10.2020]).
– **Are the developers readily accessible?** When using software, it often turns out that functionalities are missing or bugs are present in the programme. This can hinder the use of the software for certain use cases or significantly worsen the results. In such cases, the availability of the developers and their feedback on support requests is a decisive criterion. It should be clearly communicated how and where bug reports can be submitted and what information is required for rapid and accurate handling. Whether the developers cultivate an active support can be seen, for example, in the issue area of the software repository (e.g. GitHub or GitLab). Have the developers there answered enquiries promptly and satisfactorily? What is the ratio of open issues to already closed ones? How long did it take for a change to be incorporated?
+ **Is the software being actively worked on?** An indication for this are regular software updates. In a roadmap developers can proactively communicate which changes are planned for the programme in the near future and which issues are to be worked on for the next release.
+ **Is it possible to support software development?** To enable external software developers to contribute to the software, a Contribution Guideline is helpful. It contains information on the circumstances under which, and how, changes to the software are accepted and integrated by third parties (e.g. https://projectacrn.github.io/latest/developer-guides/contribute_guidelines.html [23.10.2020]).

Quality of implementation

The quality of the implementation influences whether and to what extent the programme is suitable for adaptation and further development, and thus also how long it is likely to persist.

– **Does the implementation reflect the state of the art?** This question usually has to be answered in relation to the field of application and requires a technical understanding of the processes within the software. Some aspects can be assessed without technical knowledge: is the application usable on many different devices (e.g. mobile phone, different operating systems, etc.)? Are outdated technologies avoided, such as Adobe Flash or Java applets in web applications? Do automatic test systems (for example on GitHub) identify security gaps in the software?
– **Is *Continuous Integration* used to ensure implementation quality?** For a developer, not only the existence and documentation of the source code is crucial, but s/he will usually also expect information about the functionality and compilability of the source code. As already mentioned in the previous sections, *Continuous Integration* can be such a quality indicator. The Continuous Integration process tests the compilability of the software after each change and can indicate the resulting status in the repository. It is a sign of well-maintained source code if it compiles in the currently released version and, if applicable, in the current development version.

Tabular key data for software

In the table below we present key data that can be listed in tabular form at the end or beginning of a review. This can provide a quick overview of the software, much like the way book reviews neutrally list title, publisher and ISBN. If necessary, this table can also be supplemented with hardware requirements if these are particularly relevant. We have chosen the GIS application QGIS as an example.

Name:	The name of the software, e.g. "*QGIS*".
Short description:	Summary of what the software does, e.g. "*Comprehensive graphical tool for spatial data processing*".
Reviewed version:	The software version used for the review, e.g. "*3.10.10 LTR*".
Platform:	(Operating) systems on which the software can be used, e.g. "*Windows, macOS, Linux, BSD, Android*".
Website:	URL where further information can be found, e.g. *'https://qgis.org*".
Licensing:	Software licence under which the software was published, e.g. "*Open Source with GNU General Public License (GPL)*".
Costs:	If applicable, regular or one-off licence fees, e.g. *"free of charge*".
Input and output formats:	The file formats the software can process, e.g. *"geodatabases*" (SpatiaLite, PostGIS, MSSQL, ...), *"web geodata services*" (WMD/WMTS, Vector Tiles, XYZ Tiles, WFS, ...), *"geo-vector data formats*" (ESRI Shapefile, Geopackage, ...), *"geo-raster data formats*" (GeoTIFF, ...), *"table data*" (CSV, TXT, ...) and other data types (for QGIS there are an unusually large number of data formats to consider, which would go beyond the scope here).

Fig. 1 Proposed tabular key data for e.g. QGIS.

Where do we go from here?

The discussion about the evaluation of research software in archaeology and its neighbouring disciplines has only just begun and no consensus has yet been worked out. Our considerations and the catalogue of questions presented here should therefore be directly understood as a stimulus for discussion and an invitation to establish a common understanding of the requirements for a software review. As noted at the outset, comments from a wide range of perspectives and disciplines are invited via GitHub or email. The catalogue is by no means set in stone, but rather open to modifications and updates by archaeologists and research software engineers in the future.

References

Anzt, H., Bach, F., Druskat, S., Löffler, F., Loewe, A., Renard, B.Y. et al. (2020). An Environment for Sustainable Research Software in Germany and Beyond: Current State, Open Challenges, and Call for Action. *F1000Research 9, 295*. https://doi.org/10.12688/f1000research.23224.1.

Bach, F., Druskat, S., Katerbow, M., Loewe, A., Seemann, G. (2019). *Herausforderungen für die nachhaltige Entwicklung, Bereitstellung und Pflege von Forschungssoftware in Deutschland.* Berlin: deRSE. https://de-rse.org/de/conf2019/talk/PVEXDH/ [23.10.2020].

Baxter, R., Chue Hong, N., Gorissen, D., Hetherington, J., Todorov, I. (2012). The Research Software Engineer. *Digital Research 2012*. https://web.archive.org/web/20180202071627/http://digital-research-2012.oerc.ox.ac.uk/papers/the-research-software-engineer/at_download/file [23.10.2020].

Bezjak, S., Clyburne-Sherin, A., Conzett, P., Fernandes, P., Görögh, E., Helbig, K. et al. (2018). Open Science Training Handbook. *Zenodo, 4.4.2018.* https://doi.org/10.5281/zenodo.1212496.

DFG (2019). *Leitlinien zur Sicherung guter wissenschaftlicher Praxis – Kodex*. Bonn: DFG. https://www.dfg.de/download/pdf/foerderung/rechtliche_rahmenbedingungen/gute_wissenschaftliche_praxis/kodex_gwp.pdf [23.10.2020].

Druskat, S., Spaaks, J. H., Hong, N. C., Haines, R., Baker, J. (2017). Citation File Format (CFF). *Zenodo, 4.11.2019.* https://doi.org/10.5281/zenodo.1003149.

forschungsdaten.info (2020). *Softwareentwicklung in der Wissenschaft.* Konstanz: forschungsdaten.info. https://www.forschungsdaten.info/themen/ethik-und-gute-wissenschaftliche-praxis/softwareentwicklung-und-gute-wissenschaftliche-praxis/ [23.10.2020].

Goedicke, M., Lucke, U. (2020). *Nationale Forschungsdateninfrastruktur für und mit Computer Science (NFDIxCS).* https://www.dfg.de/download/pdf/foerderung/programme/nfdi/nfdi_konferenz_2020/nfdixcs_abstract.pdf [23.12.2020].

Helmholtz Open Science (2019). *Forschungssoftware.* Bremenhaven: Alfred-Wegener-Institut, Helmholtz-Zentrum für Polar- und Meeresforschung. https://os.helmholtz.de/open-science-in-der-helmholtz-gemeinschaft/forschungssoftware/ [23.10.2020].

Hettrick, S. (2016). *A not-so-brief history of Research Software Engineers.* Edinburgh: Software Sustainable Institute. https://www.software.ac.uk/blog/2016-08-17-not-so-brief-history-research-software-engineers-0 [23.10.2020].

Hettrick, S., Antonioletti, M., Carr, L., Chue Hong, N., Crouch, S., De Roure, D., et al. (2014). UK Research Software Survey 2014. *Zenodo, 4.12.2014.* https://doi.org/10.5281/zenodo.14809.

Jung, H.-W., Kim, S.-G., Chung, C.-S. (2004). Measuring software product quality: A survey of ISO/IEC 9126. *IEEE software, 21*, 88-92.

Katerbow, M. & Feulner, G. (2018). Handreichung zum Umgang mit Forschungssoftware. *Zenodo, 27.2.2018.* https://doi.org/10.5281/zenodo.1172970.

Katz, D.S., Niemeyer, K.E., Smith, A.M., Anderson, W.L., Boettiger, C., Hinsen, K. et al. (2016). Software vs. data in the context of citation (No. e2630v1). *PeerJ Preprints, 10.12.2016.* https://doi.org/10.7287/peerj.preprints.2630v1.

Lamprecht, A.-L., Garcia, L., Kuzak, M., Martinez, C., Arcila, R., Martin Del Pico, E. et al. (2019). Towards FAIR principles for research software. *Data Science, 3(1)*, 37-59. https://doi.org/10.3233/DS-190026.

Löffler, F. (2020). *Nationale Forschungsdateninfrastruktur für wissenschaftliche Software (NFDI4RSE).* https://www.rse4nfdi.de/de/index.html [23.10.2020].

Scheliga, K., Pampel, H., Bernstein, E., Bruch, C., zu Castell, W., Diesmann, M. et al. (2017). Helmholtz Open Science Workshop „Zugang zu und Nachnutzung von wissenschaftlicher Software". *#hgfos16: Report Nov. 2016.* https://doi.org/10.2312/lis.17.01.

Schmidt, S. C. & Marwick, B. (2020). Tool-Driven Revolutions in Archaeological Science. *Journal of Computer Applications in Archaeology, 3*, 18-32. https://doi.org/10.5334/jcaa.29.

Stallman, R. (2001). *What is Free Software? The Free Software Definition*. https://www.gnu.org/philosophy/free-sw.html.en [23.12.2020].

Wilkinson, M.D., Dumontier, M., Aalbersberg, Ij.J., Appleton, G., Axton, M., Baak, A. et al. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data, 3*, 160018. https://doi.org/10.1038/sdata.2016.18.

Acknowledgements

We thank Lutz Schubert for fruitful discussions, layouting and proofreading the initial German version of this article. Furthermore, we thank Dr. Sarah Finlayson for her swift assistance in proofreading the English version, which was partially prepared using the DeepL (https://deepl.com) online translation service.

About the authors
(in alphabetical order)

Timo Homburg, Hochschule Mainz, studied computer science with a focus on computational linguistics (assyriology), semantic web and sinology. In recent years, he has worked in the field of GIS applications in geoinformatics. His doctoral thesis deals with better integration of geodata into a Semantic Web environment. His current research project deals with the design and establishment of digital standards in the field of assyriology and a best practice for documentation of excavations with cuneiform texts – an extension of his publications on 'Cuneiform Script in the Digital Humanities'.

https://orcid.org/0000-0002-9499-5840

Anne Klammt, Deutsches Forum für Kunstgeschichte Paris, completed her doctorate on a landscape archaeology topic and has been responsible for the further development of Digital Art History at the German Forum for Art History Paris as Head of Research since 2020.

https://orcid.org/0000-0003-3697-9241

Hubert Mara, mainzed & Hochschule Mainz, studied computer science at the Vienna University of Technology. There he already worked on digital methods in archaeology for ceramic analysis. He was subsequently a Marie Curie Fellow at the University of Florence within the framework of the *Cultural Heritage Informatics Research Oriented Network* (CHIRON). He completed his doctorate at the University of Heidelberg in the *Interdisciplinary Centre for Scientific Computing* (IWR). There he developed the *GigaMesh Software Framework* (https://gigamesh.eu) and installed the *Forensic Computational Geometry Laboratory* (FCGL). Since June 2020, he has been the managing director of mainzed and is a researcher at the Institute for Spatial Information and Measurement Technology at Mainz University of Applied Sciences.

https://orcid.org/0000-0002-2004-4153

Clemens Schmid, Max-Planck-Institut für Menschheitsgeschichte Jena, studied prehistoric, historical and scientific archaeology as well as computer science in Tübingen and Kiel. After graduating, he worked with computational data analysis in various archaeological research projects at the University of Kiel, the Roman-Germanic Central Museum in Mainz and the University of Bern. Now he is employed for a PhD project on quantitative estimation of past human mobility with ancient genetic and historical-linguistic data at the Department of Archaeogenetics of the Max Planck Institute for the Science of Human History in Jena.

https://orcid.org/0000-0003-3448-5715

Sophie Charlotte Schmidt, Deutsches Archäologisches Institut, studied Classical Studies and Prehistoric Archaeology at the Free University of Berlin. She then worked as a research assistant in the field of archaeoinformatics at the universities of Cologne and Bonn, and is currently employed in the NFDI4Objects consortium project at the German Archaeological Institute.

https://orcid.org/0000-0003-4696-2101

Florian Thiery, Römisch-Germanisches Zentralmuseum – Leibniz-Forschungsinstitut für Archäologie, studied geodesy and is a *Research Software Engineer* (RSE) in the *Scientific IT, Digital Platforms and Tools* department at the Römisch-Germanisches Zentralmuseum in Mainz. He is the initiator of the *Research Squirrel Engineers*, a network for RSEs in the Digital Humanities that pursues community projects with a focus on free and open data, as well as Linked Open Data. As part of a fellowship of the *Wikimedia Fellow-Programm Freies Wissen*, he is working on the modelling of Irish (Ogham) stones and their publication in the Wikimedia universe.

https://orcid.org/0000-0002-3246-3531

Martina Trognitz, Austrian Centre for Digital Humanities and Cultural Heritage, ÖAW Wien, studied computational linguistics and classical archaeology in Heidelberg. From 2012 to 2017, she worked in the IANUS project at the German Archaeological Institute. Since 2017, she has been in charge of the digital archive ARCHE of the ACDH-CH at the Austrian Academy of Sciences. She is working on a PhD project on the machine analysis of multi-sided Aegean seals of the Bronze Age.

https://orcid.org/0000-0003-0485-6861

Martina Trognitz M.A.
Austrian Centre for Digital Humanities
and Cultural Heritage
ÖAW Wien
Martina.Trognitz@oeaw.ac.at