Criteria for Reviewing Tools and Environments for Digital Scholarly Editing, version 1.0

Anna-Maria Sichani and Elena Spadini, in collaboration with the members of the IDE; Version 1.0, December 2018.[1]

> Any research tool project that is closed
> may not survive the natural curiosity of researchers.
>
> *Rockwell and Bradley 1998*

0. Preliminary remarks

0.1 Scope

0.1.1. Definitions. This paper provides a framework for the description and evaluation of digital tools and environments for scholarly editing.

The term *tool* is broadly defined as a computational application to be used for accomplishing one task or a number of related tasks in the process of digital scholarly editing. An *environment* is defined as a platform on which various tools (independent or not) are working as an integrated whole in order to perform a series of tasks or to cover the entire workflow.

In what follows, we refer to both tools and environments with the term *tool*.

We define *tool* as a piece of software or a software component: it may be web-based (web-portal or application, including web-services, web-resources and APIs) or stand-alone; users may interact with it via a Graphical User Interface (GUI), a Command Line Interface (CLI) or use it in combination with other code components (e.g. in the case of a widget or a library).[2] A tool may also be referred as a (computer) program, a (software) application or other specific denominations. Virtual Research Environments (VRE), web applications for asynchronous collaboration among researchers, are also of interest.

Digital scholarly editing is defined as the process of creating a digital scholarly edition (DSE).[3] Common tasks in this process are: collection of witnesses (document/image/metadata management and digitization), transcription, encoding, named-entity recognition, semantic enrichment, collation, analysis, constitution of critical (or copy) text, compilation of apparatus, compilation of indexes, preparation of paratextual material, data visualization, and publication. Although this list is quite long and diverse, it is not and it could never be comprehensive, especially as the above-mentioned tasks are not firmly restricted to digital scholarly editing endeavours, and given that technology and the field itself are constantly evolving. Depending on the materials, the aims and the methods, a

number of procedures can be applied to the materials to be edited. For instance, NLP or text analysis techniques, which have not been mentioned here, might be relevant.

For this reason, we consider *a tool for digital scholarly editing* any piece of software that can be used in this process. The tool in question might be originally developed for or within a digital scholarly editing workflow or be a generic tool repurposed for such process. Depending on the commonality of the task in relation to digital scholarly editing, the author of the review should provide adequate justification for its inclusion. For example, a tool for text encoding is more frequently linked to editions than a tool for topic modeling; thus in describing and evaluating the second, the reviewer should provide additional information about its significance and application with respect to digital scholarly editing.

0.1.2. Traditions and context

There is an ongoing and vivid discussion about *tooling*, defined as the process of creating a tool or of working with tools,[4] in Digital Humanities. In general, DH practitioners tend to agree that tools play an important role in the scholarly ecosystem,[5] while acknowledging different issues related to their development, design and application.

Probably the first conference on tools for textual scholarship was organized by Susan Hockey at Princeton in 1996. The critical features identified at the Text Analysis Software Planning Meeting are still key issues today: modularity, professionality, integration, portability (Sperberg-McQueen 1996).

The history of text-oriented applications swings between tools developed for specific uses and general-purpose ones: John Unsworth (2003) distinguished four generations, from *ad hoc* programs, through reusable libraries of text-processing routines, to interactive general-purpose programs, which operate via the web in the last phase. The same tension between specific and general-purpose tools, entailing limitations and ease of use *vs* complexity, is described as the “tool-paradox” and addressed in terms of future development by Pape et al. (2012).

The question of how to build “better” tools has been frequently posed. Among others, Juola (2008) investigated the reasons for the low rate of adoption of DH tools in the Humanities and proposed strategies for building “killer applications”, filling the needs of the user communities. Recently, Dennis Tenen (2016) advocated that “[t]he tool can only serve as a vehicle for methodology”; therefore, methodological development should be prioritized in order to sharpen the tools. Awareness in the use of tools is the first step towards this end: “[t]o use such tools well, we must, in some real sense, understand them better than the tool makers. At the very least, we should know them well enough to comprehend their biases and limitations” (Tenen 2016). The question of bias and constraints is addressed from a different angle by Stephan Ramsey (2016), considering those as key elements that can lead to thorough analysis: “thoughts and discussions are the result of the constraints, not something that happens as a side effect”, or more plainly, “[t]he creator of a visualization is likewise trying to constrain the user’s view by *limiting* possible interpretations, so that some other feature can be more easily seen”. Ramsey’s reflection starts from the assumption that “digital tools and frameworks are based upon a fundamental set of primitives (algorithms and data structures)”.

Another aspect of the discussion is about the value and the related credit to be assigned to tools and tooling. In the first issue of the *Debates in the Digital Humanities* series (2012), Ramsey and

Rockwell, developers in Digital Humanities themselves, argued that, despite the widespread anxieties about credit for digital work and an undeniable resistance in certain parts of academia caused by the "fear of an automated scholarship", "it would still fall to the builders to present their own activities as capable of providing affordances as rich and provocative as that of writing" (Ramsey and Rockwell 2012).

In the field of digital editing, some scholars have expressed a comprehensive approach towards the nature and the role of tools within the scholarly research framework, for instance proposing to build editions as "composites of three types of digital web services: data sources, processing services, and interfaces" (Boot and van Zundert 2011; see also van Zundert 2018), or to consider them as resources that "necessarily comprise all three components of a digital publication — the source, the output and the tools to produce and display it — and it is worth emphasizing again that all three are scholarly products that result from editorial practice" (Pierazzo 2015). In Schmidt's somewhat provocative definition, "a digital scholarly edition is software" and should therefore "be governed by the procedures and principles of software development" (Schmidt 2014).

Both in the field of Digital Humanities and elsewhere, scholars have defended the view that digital tools and software are research outputs and should be treated as full and equal parts of the scholarly ecosystem (Crouch et al. 2013). Pushing in the same direction, others have provided critical investigation of the current situation (Schreibman and Halon 2010). Recently, awareness of the scholarly value of software products has increased — originally in STEM (Science, Technology, Engineering and Mathematics) disciplines — thanks to a number of initiatives, such as dedicated journals for publishing and reviewing software (The Journal of Open Source Software, SoftwareX) and guidelines such as those produced by the Software Sustainability Institute and CLARIAH (NL). The above-mentioned initiatives aim to promote quality software development and assessment, along with the research software engineer career path, and software as part of the research agenda. A different but related goal is pursued by Software Heritage, committed to collect, preserve and share publicly available software in source code form.

0.1.3. Purpose. In line with such initiatives, these Criteria aim to further raise and promote awareness towards the (autonomous and original) research value of computational tools and software in the digital scholarly editing ecosystem, and more broadly in Digital Humanities research. To this end, we have developed a set of criteria for quality assessment of the digital tools currently available or previously used for digital scholarly editing.

By reviewing not the final scholarly resource (i.e. the digital scholarly edition itself) but the usually hidden steps that precede this — while essentially enabling it — we aim to further problematize the current academic distinction between tools and products/results. As Andrews (2013) states: "the method of production, rather than the published form that the resulting edition take, is the practice wherein lies most of the promised revolution within textual scholarship".

Furthermore, these Criteria aim to encourage and support the development of digital research tools and software of high quality according to a set of best practices. Despite the objective differences in terms of governance model, resources and aims, between industry and academia and between commercial and public sectors, if we accept that tool development and usage is part of the scholarly research ecosystem, we must also propose, debate and integrate relevant procedures for quality assessment.

Finally, by these Criteria we want to provide a list of the currently available computational tools for digital scholarly editing and to assist the process of choosing the suitable digital tool during a digital scholarly editing project.

0.2. Application

0.2.1. General and specific criteria. A number of general criteria are applicable to most tools or similar resources. All the relevant criteria must be addressed in the review in their entirety. On the other hand, there are specific and detail-oriented criteria, which can be applied only to certain kinds of tools. It falls to the reviewer to identify them.

0.2.2. Parameters. Tools should be reviewed on a case-by-case basis considering their individual features: goals set by the creators, financial resources, the run-time of the project and other available resources. It might be appraised positively when a tool is very rich in scope and functionalities. But it must also be recognized when only narrower, self-defined aims are met. Where the defined goals are not fulfilled, the stage of development should be taken into account. Publication and review are often valid only momentarily in an ongoing, open process of scholarly engagement, which should not be mistaken for the final, completed state of a tool. Therefore, the reviewer must examine whether the tool has, at the point of its review, reached sufficient maturity and consistency to be a worthwhile resource for research. See also section [*1.3. Stage of development*].

0.2.3. Qualification of the reviewer. Reviewing tools for digital scholarly editing requires a double qualification, because the reviewer has to be able to evaluate the technical aspects of a tool and its usefulness as a research instrument in the context of a specific discipline or a community of practice. Reviews of tools may only focus on one of these sides, which must be stated explicitly. Ideally, a complementary opinion by experts from fields of study in which the reviewer has less expertise should be added to the text. See also section [*1.1. The reviewer*].

0.2.4. Requirements. Reviewers are expected to have a good knowledge of the tool and to have used it in a systematic way with an appropriate use case.

1. Opening the review

1.1. The reviewer. If relevant to the review, provide your academic background, prior experience, and research interests. This can be done in an extensive footnote. The reviewer should specify if s/he knows the tool as a user, a user-developer, or a developer (Jackson et al. 2011). A user is a person who downloads, installs, configures and uses the tool under review but does not necessarily write any code to use in conjunction with it. The user-developer is a user who writes code that extends but does not alter the core functionality of the tool under review. A developer is a user who writes code that changes the tool under review, e.g. fixes bugs, makes the software more efficient, or extends its functionality.

1.2. Identification of the reviewed tool. The identification of the tool should be as specific as possible (Smith et al. 2016). It should include a unique identifier (see below), an active access link, the full tool’s name and any available acronyms, the author(s), the version number, the revision number, the release date, the storage location or repository. If some metadata are not available, alternatives may be provided: the download date can substitute the version number and release date;

the contact name/email is an alternative to the location/repository. A unique identifier refers to unique, persistent, and machine-actionable identifiers such as a DOI, ARK, or (P)URL.[6]

1.3. Stage of development. A tool can be ready for production, in alpha or beta phase. Obsolete or legacy tools are not intended to be reviewed although they might be of historical value for the development of the field.

1.4. Identification of the reviewing platform. The environment in which the tool is executed must be specified. This may be an operating system, a web-browser, another application, or a virtual machine.

1.5. General introduction. This section serves as a first presentation of the tool: all aspects touched upon will be detailed in the following sections of the review. The scope of the tool should be briefly described, including what the tool does, which task it fulfils and what desiderata it addresses. It should also be specified if the tool has been developed for general purposes or for a particular project. A further distinction can be made between end-user tools (for production) and experimental tools (proof-of-concept).

1.6. Credit and funding. Indicate who are the researchers, institutions, funders and other entities involved in the creation of the tool, and what are their responsibilities. Researchers may include academic researchers, research software engineers, developers, etc. Relations to other projects should be mentioned if relevant. If this information is available, list the financial, personnel and time resources available considering the governance model (that is, how the project is implemented). The financial model of the tool should also be discussed, i.e. if its use is free/open, support-based or entirely behind a paywall.

1.7. Transparency. Specify if the general parameters are easily accessible, in which form (e.g. readme file, .txt, .html, .pdf), if contact information is present and if credit is adequately given. See also section [*3.5. Source, license and credits*].

2. Methods and Implementation

2.1. State of the art. Provide information about the state-of-the-art: how does the tool relate to other digital tools and resources, to its predecessors or to similar projects.

2.2. Discipline and methodology. A tool makes possible some form of processing, putting into action a scientific choice. Unveiling the methodological approach enabled by the tool is one of the aims of the review. The methodology is related to the discipline, area or task for which the tool is originally designed and/or in which it is used, and can also be trans-disciplinary. The “potential to impact the process of scientific discovery” should be expressed (SoftwareX 2015).

2.3. How it works. A description of how the software accomplishes its task and of its main features is required. This may be accompanied by a visual map or a drawing offering an architectural overview of the software and/or a flowchart of the program logic (algorithm). Links to relevant publications should be listed. Mention the programming languages and the technologies used, and if the tool reuses portions of other existing software. If possible, specify the reasons for the technical choices (e.g. as a trade-off between local conditions and best practices).

2.4. Input, output, and data-modelling. Specify the input and output formats (e.g. .xml, .txt, .pdf, etc.) and encodings (e.g. latin-1, utf-8) supported. Point out if the internal operations require a form

of data modelling or conversion, e.g. if schema control is present or pre-processing format conversion is included. With regards to multilingual audience, mention whether the tool is compatible with textual resources using non-Latin alphabet/scripts and/or multiple languages.

2.5. Performance. Performance can be assessed by looking at accuracy and efficiency. Accuracy refers to whether the tool delivers what it was built for; efficiency indicates whether the tool does it in time and with respect to the use of resources, that is if it employs an efficient algorithm. Note that there is often a trade-off between speed and memory usage. If the software is intended to handle big-data or large user bases, performance should remain acceptable when doing so. Performance also influences the responsiveness of the interface, for example if the interface is blocked while the tool is performing a task, or if on the contrary the tool uses asynchronous methods for handling user-interface interaction.

2.6. Logs and trackability. Indicate whether the tool offers logs, versioning or other ways to record the work done, for debugging and for the reproducibility of the results obtained (cf. Rockwell and Bradley 1999).

2.7. Application and results. Indicate if the tool is in use in any research project, with particular attention to DH projects and to digital scholarly editing. Consider the applications and the results obtained with the tool in these projects.

2.8. *Example*. An example of how the tool works is not required but strongly welcome. A suitable use case, e.g. the completion of one task, should be chosen and it is recommended to provide explanatory notes.

3. Usability, sustainability and maintainability [7]

3.1. Access. Mention from where the tool can be downloaded or acquired, e.g. if it is deposited in a public repository, such as a programming language or platform repository[8], or a repository intended for research software publication.

3.2. Dependencies. Specify if the documentation lists as dependencies other software, libraries or hardware, and if they should be installed before the installation of the tool or if they are handled by the installation process itself.

3.3. Documentation. Indicate the existence of a tool website, wiki, blog, documentation, or tutorial. Specify if documentation is accessible (e.g., also from the interface through a ‘Help’ option) and in which format (readme, .txt, .html, .pdf, etc.); if it is clear, well-structured and accurate; and if it is up-to-date (using versioning). Specify also if the documentation includes and clearly covers: a ‘Getting Started’ section (installation and configuration), step-by-step instructions, examples, troubleshooting (a selection of possible error messages and related solutions), a FAQ or a support section, and an API documentation if relevant.

3.4. Support. Indicate if active support from the developer(s) or from the community is available through a help desk, a forum, a mailing-list or other; and if it is possible to post bugs or issue using issue tracker mechanisms.

3.5. Learnability. Evaluate the learning curve of the tool, considering criteria such as knowledge requirements („what I need to know to make it work“, as opposed to technical requirements like dependencies) and learnability of the interface. For the latter, take into account the visibility and the

location of the functionalities, the continuity of task flows, the error and feedback messages. Provide an overall evaluation of the learning curve, based on your experience.

3.6. Build and install. Express how straightforward it is to build or install the tool on a supported platform.[9] Indicate, for instance, the presence of a single command to build the software, and to install or uninstall the software.

3.7. Test. Consider if there is a test suite, covering the core functionality in order to check that the tool has been correctly built or installed. The result of the test can be used to assess the accuracy of the tool (see section *2.5. Performance*).

3.8. Portability and interoperability. Portability indicates the ability of the tool to be used on multiple platforms (Linux/BSD/Unix, Mac OS X, Android, iOS, Windows), devices (desktop, mobile, etc.) and/or browsers (Mozilla Firefox, Google Chrome, Safari, etc.). For web-based tools, specify if the tool relies on browser plugins and if they are themselves portable. Interoperability indicates the ability to work in conjunction with other software, typically by support for multiple open-standard formats as input and output.

3.9. Source, license and credits. Indicate if the source-code is open and under which license the tool is distributed. Specify if this information is accessible and where. If available, mention if the tool is hosted in a public version-controlled repository and where development takes place. Address whether adequate acknowledgement and credit is given to the project contributors.

3.10. Analysability, extensibility, reusability of the code. The source code may be inspected in closer detail by the reviewer if s/he has the necessary skills. In this case, consider one or more of the following: the analysability of the code (structure, comments, naming, standards), its extensibility (contribution mechanisms, attribution for changes, backward compatibility) and its reusability in other contexts (appropriate interfaces, modular architecture).[10]

3.11. Security and privacy. Indicates if sufficient information is provided about the treatment of the data entered by the users.

3.12. Supportability and maintenance. If possible, mention whether the tool will be supported currently and in the future and under which framework/status.

3.13. Citability. Indicate if the tool supplies citation guidelines, e.g. using the Citation File Format.[11]

4. User interaction, GUI and visualization

4.1. User profile. Define what kind of user and interactions from the user are expected, and to whom the tool is addressed. See also section [*1.1. The reviewer*].

4.2. Technical interfaces and GUI. It should be specified if the tool comes with a standard Graphical User Interface (GUI) or if it is offered as a bundle of files available for download. If a GUI is present, or in the case of a web-based tool, relevant elements of the interface design should be highlighted. Mention, for instance, if the GUI is clearly arranged, so that the user can quickly identify the different operations available and their concatenation in a workflow; if the GUI is in line with common visual patterns; if the tool status is visible; if there is the possibility to search in the data and in the tool functions. Indicate the possibility of customizing the GUI, e.g. hiding

irrelevant graphical control elements (menus, toolbar, dialog boxes, etc). Specify if the GUI provides mechanisms for error prevention, through warnings, “undo” buttons and error messages. In case a GUI is not present, some of these characteristics are addressed in section [*3.3. Build and install*]. Further details can be provided about how straightforward it is to interact with the tool, for instance in the organization of commands, functions and related parameters.

4.3. Visualization. Indicate if the tool provides particular visualizations of the input and/or the output data. If yes, specify if the visualizations are self-explanatory and, if not, if they are accompanied by explanations and/or interpretation; if they are helpful in the context of the operations available; and if they lead to new research insights.

4.4. User empowerment. Mention to what extent the user is allowed to customize the functioning of the tool and the output configuration, building her/his own workflow.

4.5. Accessibility. Indicate if the tool provides particular features for improving accessibility, allowing “people with the widest range of characteristics and capabilities to use it. The range of capabilities includes disabilities and impairments associated with age” (ISO/IEC 25010:2011).

5. Conclusion

5.1. General characteristics and realisation of aims. According to the descriptions outlined in these Criteria, the reviewer must present the main characteristics of the reviewed tool and whether or to what extent the tool has successfully accomplished its original aims.

5.2. Achievements and scholarly contribution. The review should clearly state to what extent the tool contributes to current scholarship and has a methodological impact.

*5.3. Particularities and limitation*s. Provide a summary of features that merit special attention for noteworthiness and/or innovation, even if they are beyond the scope of these Criteria, and of the limitations of the tools.

5.4. Suggestions for improvement. If the project is not complete and finished, mention what should be considered for further improvement, and what useful additions can be made. If the project is terminated, express what would be the most desirable steps for extending it, if any.

6. Works cited

Andrews, Tara. 2013. “The Third Way: Philology and Critical Edition in the Digital Age.” *Variants* 10: 61–76.

Bleier, Roman, Martina Bürgermeister, Helmut W. Klug, Frederike Neuber, and Gerlinde Schneider, eds. 2018. *Digital Scholarly Editions as Interfaces*. Schriften des Instituts für Dokumentologie und Editorik 12. Norderstedt: BoD. http://kups.ub.uni-koeln.de/9085/

Boot, Peter, and Joris van Zundert. 2011. “The Digital Edition 2.0 and the Digital Library: Services, Not Resources.” (Digitale Edition und Forschungsbibliothek. Beiträge der Fachtagung im Philosophicum der Universität Mainz am 13. und 14. Januar 2011). *Bibliothek und Wissenschaft* 44: 141–52.

Bradley, John. 2002. “Tools to Augment Scholarly Activity: An Architecture to Support Text Analysis.” In *Augmenting Comprehension Digital Tools and the History of Ideas*, edited by Harold Short, Dino Buzzetti, and Guiliano Pancaldi, 19–48. Office for Humanities Communication.

Burdick, Anne, Johanna Drucker, Peter Lunenfeld, Todd Presner, and Jeffrey Schnapp. 2012. *Digital_Humanities*. Cambridge, MA: The MIT Press. https://mitpress.mit.edu/books/digitalhumanities.

CLARIAH. 2016. “Guidelines for Software Quality.” Task 54.100. https://github.com/CLARIAH/software-quality-guidelines/blob/master/softwareguidelines.pdf.

Crouch, Stephebn, Neil Chue Hong, Simon Hettrick, Mike Jackson, Aleksandra Pawlik, Shoaib Sufi, Les Carr, David De Roure, Carole Goble, and Mark Parsons. 2013. “The Software Sustainability Institute: Changing Research Software Attitudes and Practices.” *Computing in Science Engineering* 15 (6): 74–80. https://doi.org/10.1109/MCSE.2013.133.

Henny, Ulrike, and Frederike Neuber, in collaboration with the members of the IDE. Version 1.0 (February 2017). “Criteria for Reviewing Digital Text Collections.” https://www.i-d-e.de/publikationen/weitereschriften/criteria-text-collections-version-1-0/.

ISO/IEC. “Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — System and Software Quality Models.” ISO/IEC 25010:2011(en). https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en.

Jackson, Mike, Steve Crouch, and Rob Baxter. 2011. “Software Evaluation: Criteria-Based Assessment.” Software Sustainability Institute. https://software.ac.uk/sites/default/files/SSI-SoftwareEvaluationCriteria.pdf.

Juola, Patrick. 2008. “Killer Applications in Digital Humanities.” *Literary and Linguistic Computing* 23 (1): 73–83. https://doi.org/10.1093/llc/fqm042.

Pape, Sebastian, Christof Schöch, and Lutz Wegner. 2012. “TEICHI and the Tools Paradox.” *Journal of the Text Encoding Initiative*, no. Issue 2 (February). https://doi.org/10.4000/jtei.432.

Pierazzo, Elena. 2015. *Digital Scholarly Editing: Theories, Models and Methods*. Ashgate.

Ramsay, Stephen. 2016. “Hard Constraints: Designing Software in the Digital Humanities.” In *A New Companion to Digital Humanities*, edited by Susan Schreibman, Ray Siemens, and John Unsworth, 449–57. Chichester, UK: John Wiley & Sons, Ltd. https://doi.org/10.1002/9781118680605.ch31.

Ramsay, Stephen, and Geoffrey Rockwell. 2012. “Developing Things: Notes toward an Epistemology of Building in the Digital Humanities.” In *Debates in the Digital Humanities*, edited by Matthew K. Gold. University of Minnesota Press.

Rockwell, Geoffrey, and John Bradley. 1998. “Eye-ConTact: Towards a New Design for Text-Analysis Tools.” *Digital Studies/Le Champ Numérique*, February. https://doi.org/10.16995/dscn.232.

Sahle, Patrick, in collaboration with Georg Vogeler and the members of the IDE. Version 1.1 (June 2014). “Criteria for Reviewing Scholarly Digital Editions.” https://www.i-d-e.de/publikationen/weitereschriften/criteria-version-1-1/.

Schmidt, Desmond. 2014. “The Digital Scholarly Edition: What Is It and How Do We Make It.” In . Rome. https://digilab.uniroma1.it/en/archivionotizie/scholarly-digital-edition-and-humanities.

Schreibman, Susan, and Ann M. Hanlon. 2010. “Determining Value for Digital Humanities Tools: Report on a Survey of Tool Developers.” *Digital Humanities Quarterly* 4 (2). http://www.digitalhumanities.org/dhq/vol/4/2/000083/000083.html.

Smith, Arfon M., Daniel S. Katz, and Kyle E. Niemeyer. 2016. “Software Citation Principles.” *PeerJ Computer Science* 2 (September): e86. https://doi.org/10.7717/peerj-cs.86.

SoftwareX. 2015. “Software Articles.” https://www.elsevier.com/authors/author-services/research-elements/software-articles.

Sperberg-McQueen, Michael. 1996. “Trip Report: Text Analysis Software Planning Meeting, Princeton, New Jersey, May 1996.” http://www-01.sil.org/cellar/import/teilite/ceth9605.sgm.

Tenen, Dennis. 2016. “Blunt Instrumentalism: On Tools and Methods.” In *Debates in the Digital Humanities*, edited by Lauren Klein and Matthew Gold. University of Minnesota Press.

The Journal of Open Source Software. n.d. “Review Criteria.” https://joss.readthedocs.io/en/latest/review_criteria.html.

Unsworth, John. 2003. “Tool-Time, or ‘Haven’t We Been Here Already?’: Ten Years in Humanities Computing.” In *Transforming Disciplines: The Humanities and Computer Science*. Washington, D.C. http://www.iath.virginia.edu/~jmu2m/carnegie-ninch.03.html.

Zundert, Joris van. 2018. “On Not Writing a Review about Mirador: Mirador, IIIF, and the Epistemological Gains of Distributed Digital Scholarly Resources.” *Digital Medievalist* 11. https://doi.org/10.16995/dm.78.

Notes

[1] These guidelines are a spin-off derived from the *Criteria for Reviewing Scholarly Digital Editions, version 1.1* (Sahle 2014) and the *Criteria for Reviewing Digital Text Collections, version 1.0* (Henny and Neuber 2017). The following parts of the *Criteria for Reviewing Scholarly Digital Editions* have been partly or entirely integrated into the present guidelines: Application; 1.1/3/4/5; 5.2/4/5/7. The following parts of the *Criteria for Reviewing Digital Text Collections* have been partly or entirely integrated into the present guidelines: 3.2, 5.3/4/6/7. Other sources are quoted in the text and recorded in the section *Works cited.*

[2] See Bleier et al. 2018 on the relationships between interfaces and the digital edition.

[3] For the definition of DSE, please refer to the *Criteria for Reviewing Scholarly Digital Editions*.

[4] Oxford Dictionary: “*Tooling*. 1.1. The process of making or working something with tools”. <https://en.oxforddictionaries.com/definition/tooling>

[5] See, for example, Burdick et al. 2012, p. 10: “The advent of Digital Humanities implies a reinterpretation of the humanities as a generative enterprise: one in which students and faculty alike are making things as they study and perform research, generating not just texts (in the form of analysis, commentary, narration, critique) but also images, interactions, cross-media corpora, software, and platforms”.

[6] Cf. CLARIAH 2016, 6.3-AC7: “Users using Github (see AC2) and Zenodo can set this up with little effort. Any releases made on GitHub will then automatically transfer to Zenodo and receive a DOI, no user intervention is necessary”.

[7] Jackson et al. 2011 and CLARIAH 2016 have been extensively used in this section.

[8] E.g. <https://pypi.python.org/pypi> for Python; <https://play.google.com/store> for Android.

[9] Building applies to software that is not interpreted at run-time from the source-code, but need to be compiled (e.g., C, C++, Java, Pascal, Haskell, Scala, Rust, Cython).

[10] For further information, see CLARIAH 2016, 6.8, 6.9 and 6.10 (please note that CLARIAH describes best practices for developing a software, which is different from reviewing it).

[11] <https://citation-file-format.github.io/>