# Exploring Curation-ready Software: Improving Curation-readiness

**Curation Ready Software Working Group report, July 27, 2017**

Fernando Rios, Bridget Almas, Alexandra Chassanoff, Nicole Contaxis, Paula Jabloner
DOI 10.17605/OSF.IO/T9G3Q

## Introduction

The Software Preservation Network's Curation-ready Software working group endeavors to develop use-case driven guidelines for improving the quality of preserved software given available resources ("curation-readiness"), including expertise, technical infrastructure, and time. This report is a continuation of Exploring Curation-ready Software: Use Cases by Rios et al. (2017)[1]. In that report, a series of software preservation use cases, spanning the archival/museum and research perspectives were presented. The purpose was to detail some of the intricacies which need to be considered when preserving software in each case. This included identifying stakeholders, outlining specific preservation goals, and outlining stakeholder responsibilities. This report aims to further expand what it means to preserve software in the context of each use case and how one might move closer to that goal by making software curation-ready.

On many levels, the problems facing the preservation of software over a long period of time are similar to that of preserving any other kind of digital information. That is, preserving digital information involves among other things (e.g., planning), preserving the bits that encode the information. However, preserving the bits that comprise source code or executables is not enough to make the software usable. In fact, even coming to an agreement on what it means to preserve software can be challenging given the different needs of the many preservation scenarios that can exist when viewed through cultural heritage or research lenses. For example, does preservation mean (current or eventual) access or reuse? To shed some light on these differences, the use cases presented in this report each highlight what it means for software to be curation-ready for that particular case and what can be done to increase its curation-readiness so that it better achieves its specific preservation objectives.

## Use Cases

These use cases are a subset of ones presented in the Exploring Curation-ready Software: Use Cases report (Rios et al., 2017)[1]. To summarize, the use cases can be associated with the research or museum/archival perspectives. Viewed from the archival/museum perspective, software is treated as a software object which has some historical, cultural, or artistic significance. On the other hand, the preservation of software viewed from the research perspective is concerned with enabling the preservation

[1] Rios, Fernando, Bridget Almas, Nicole Contaxis, Paula Jabloner, and Heidi Kelly. 2017. "Report: Exploring Curation-Ready Software: Use Cases." Open Science Framework. May 16. doi:10.17605/OSF.IO/8RZ9E.

of a scholarly body of work for the purposes of understanding, reproducing, and reusing part of, or the entire body of work.

These use cases were drawn from the author's own area of expertise in regards to software and therefore reflect each of the individual experiences and pain points encountered when dealing with software preservation issues. Cases A1 to B2 are closely associated with the research perspective and therefore reflect many of the difficulties associated with software development and preserving the scholarly record. The remaining cases are more closely associated with the museum/archival perspective and reflect the relevant difficulties such as collecting associated material, organizing it, and ensuring software remains accessible and, if possible, usable.

To facilitate comparisons across use cases, each case is described using the same structure. That is, each case begins with a short description, followed by a short summary of salient qualities. These first two two items summarize the information from Rios et al. (2017). The current report goes further to then present a definition for what it means for the software to be curation-ready in the context of that use case. This is followed with a description of the kinds of issues, questions, needs, etc. that should be addressed to increase the software's curation-readiness. Finally, a list of things that a curator, archivist, or research data manager could do to address the issues, questions, and needs is presented.

# Case A1

This use case deals with software developed as part of the research process, either as a tool that supports addressing research questions of interest (e.g., data processing and analysis scripts, figure generation, or data collection software) or as the result of the research itself (e.g., implementation of novel algorithms, decision support tools, tools to enhance other people's research, etc).

**Salient qualities of use case**

- Produced in the course of academic research (usually, but not necessarily, scientific research)
- Enable research scrutiny, reproducibility, and reuse; preserve the scholarly record
- Source code openly released
- Curation takes place at the end of development

**Curation-readiness definition**

A software artifact (i.e., executables or code) that is ready to be shared, found, understood, cited, compiled and/or executed, and preserved to ensure it remains buildable/executable, along with adequate metadata which supports the intent of preservation.

**How to increase curation-readiness**

The most important quality that affects the curatability of software in this case is the fact that the research has ended and changes to the code are no longer possible due to various reasons (funding has ended, relevant personnel have left, etc).

To increase the curation-readiness of code, the most important questions revolve around determining the purpose of preservation as this will have implications for the approaches and effort needed to achieve the preservation goals. For example, preserving code so others can examine it will require less effort than preserving the ability to re-execute the software. The metadata needed to compile the software may overlap, but not be identical to the metadata needed to find and cite it.

For citation purposes, it is sufficient to include the title of the software, the authors, version, location where the software can be found, and a unique identifier such as a DOI. For discovery, keywords and a brief description are additionally needed. For understanding, using and modifying the code, more comprehensive information is needed such as a user's or developer's manual, code comments, and critically, information on which dependencies are needed and how to obtain them. For evaluating the trustworthiness of the software, information on who else is using the software, where it has been cited, and how it has been validated is very useful.

To make a software package more curation ready, packaging it and describing it at a level of granularity that is appropriate for the preservation and/or sharing goal is very useful. For example, if a software has multiple parts and the audience is developers, packaging and describing the relevant parts individually might be the more useful approach. On the other hand, if the audience is end users, collecting everything into a single package and describing it as a whole would be more appropriate.

Finally, it is important to remain mindful of intellectual property issues. Communicating how a piece of code can be used, modified, and (re)distributed is a critical part of sharing software. Explicitly attaching a license or other terms is the way this information is best communicated. There are also other issues that should be kept in mind such as meeting funder requirements for releasing software and potential pitfalls such as adhering to export restrictions (e.g., cryptographic software).

**How can curator/curating body help? (Actions, Q's, criteria, etc.)**
The following actions would help increase the curation-readiness of research software that has been already developed.

*Elicit the purpose of the software and goals of preservation/sharing.* This will help in identifying the preservation and sharing approach (e.g., how to package the software, where to put it, etc.). To elicit this information, questions that can be asked include: "how do you envision your software being used by others?" or "do you envision others repurposing parts of your code?"

*Help with license selection.* Attaching a license is how terms of re-use are communicated. If no license is attached, standard copyright applies even if source code has been released (at least in the US) which may unintendedly hinder reuse.

*Help with determining at what level of granularity/detail to describe and package the software.* This is important as different audiences and reuse scenarios may benefit from packaging and describing the software as a whole or as individual pieces that can be separately reused. Determining this granularity is informed by the preservation and sharing goals mentioned above.

*Suggest the use of virtual machines or containers if the intended use case would benefit.* This is related to the granularity/detail activity above. For example, complex software with many dependencies, difficult configuration, or a desire for longer term reproducibility may benefit from packaging in a virtual machine. Note that virtual machines and containers should NOT be the only thing that is preserved since disentangling the software from the virtual machine environment may be difficult and may hinder reuse in new scenarios.

*Be prepared to harvest most information without the software creator's help.* Because it can be difficult to convince researchers to provide structured metadata, curators should be prepared to collect and structure this metadata using any documentation that was actually provided, code comments, websites, literature searches etc. This harvested information can simply be presented to the researcher that wrote the software for their approval.

# Case A2

This case is nearly identical to A1 with the key difference that the software is still early in development. This means that more can be done to plan for and adopt practices which make the software more easily understandable, shareable, and preservable, the end goal being reusability over time.

**Salient qualities of use case**

- Same as Case A1
- Curation activities can take place before starting and during the software development lifecycle as well.

**Curation-readiness definition**

Same as Case A1

**How to increase curation-readiness**

In addition to what was outlined for Case A1, there are several ways to increase curation-readiness which arise from established software development best-practices. Refer to reproducible research best practices[2] for details The overarching theme to these recommendations is that development should take place assuming the software will be shared (even if it actually won't be). Specific actions and recommendations are outlined below.

**How can curator/curating body help? (Actions, Q's, criteria, etc.)**

In addition to what was outlined for Case A1, the following actions can be taken to increase the curation-readiness of software that is in the early stages of development.

*Recommend use of version control systems.* Version control systems encourage a more robust software development process and allows for more easily maintaining and improving the code (e.g., easier to find and fix bugs). It also makes it easier to identify which version of the software was used to obtain a specific research output. This is useful for citation and reproducibility.

[2] E.g., http://uwescience.github.io/reproducible/guidelines.html or https://esipfed.github.io/Software-Assessment-Guidelines/

*Recommend a mechanism for dependency tracking.* Software dependencies (i.e., modules a particular piece of code depends on) can greatly complicate the sharing and preservation of software. Something as simple as listing the names and version numbers of any software libraries that a particular software requires can be of great help to those that will reuse it. For more complex codebases, making use of dependency management tools like Maven or package managers and installation scripts like those used with languages like Python or R can be highly beneficial for users of the code. Keeping copies of dependencies may not be needed unless those dependencies have a high chance of becoming unavailable during the preservation time frame.

*Recommend open or non-proprietary libraries, platforms, data formats.* This is related to the previous point on dependencies. Even if dependencies are listed, if they cannot easily be obtained, the problem of reusing someone's work becomes more complicated. By encouraging the use of non-proprietary tools and data formats, it becomes easier to preserve a complete, nearly self-contained software artifact that can more easily be used by others. A resource that lists a wide variety of open source libraries and frameworks is Libraries.io.[3]

*Curator should be knowledgeable on basic software development best-practices.* Basic things like documenting the purpose of the code, what is needed to compile and run it, and how it should be used are critical pieces of information that should be included with the code. Some code-specific best-practices include

- Eliminating dependencies on specific hardware, operating systems, or file structures, when possible. For example, it is not recommended to use direct system calls as it makes the code less portable to other systems. It is also not recommended to hardcode file paths into source code. Instead make the paths to folders or files a parameter that can be given by the user.
- Including example data and tests when possible. This allows the verification of the software's functioning.

*Recommend code be modular and structured so that specific statements in a paper can be linked to specific parts of the code.* This makes it easier for others to understand how specific claims or results arise. Alternatively, the curator should be prepared to make a recommendation on workflow management and capture tools if appropriate (e.g., Taverna, VisTrails, Reprozip, and many others) and how code can be developed to integrate with those tools.

# Case B1

This use case deals with the curation of software that is part of a network of distributed services and linked data and supports an online digital publication of a scholarly work product. The software itself is not the primary research output but the research product is dependent upon it for its presentation. Development is either not started, is at an early stage, or has at least some continued funding to support preservation activities.

[3] https://libraries.io/

**Salient qualities of use case**

- Research product is dependent upon it for presentation
- Part of a larger, service-based ecosystem
- Many external dependencies
- Development is funded and is not started or in early stage

**Curation-readiness definition**

The software would be deemed to be curation ready if it can be distributed as a package that can be shared, whether for preservation and archiving, or reuse. It would contain documentation on installation/deployment, use and architecture/design; clear identification of any dependencies (with dependencies included if possible), unit tests, and identification of all contributors to the development of the software, including individuals who participated in its design and testing as well as the actual coding.

**How to increase curation-readiness**

While all of the recommendations in the prior use cases, A1 and A2, are pertinent to this use case, the complex nature of distributed software necessitates some further considerations that should be taken into account from the start of the software development process.

- Design the code so that all dependencies on external services and linked data sets are configurable characteristics that must be defined in order to deploy the software. In combination with the use of a configuration management tool, as discussed in further in the next point, this will enable you to make sure that all dependencies are clearly documented. In addition, make sure that external service and data dependencies are also identified in the user interface of the software to ensure that sources are properly cited and credited.

- To facilitate packaging and redistribution of the software, from the very start of development use an open source configuration management tool (such as Puppet[4]) to deploy the software. This will ensure that all deployment dependencies are exposed and documented via the configuration management tool manifests and that the act of deploying is a consistently repeatable process.

- Future proof the software against changes in the external libraries on which it depends. Make sure explicit versions of all 3rd party library dependencies are defined in software packaging manifests, rather than using default or latest versions. Additionally, as the required versions of external and 3rd party libraries and services may not be guaranteed to be available in perpetuity, creating a local repository of all 3rd party library dependencies can be a good practice.

- Evaluate sustainability policies and long-term viability of external services and data set dependencies prior to using them. In addition, consider creating local snapshots of any data sets upon which the software or publication depends, and and provide a configurable switch in the code to enable use of the snapshot rather than the live data.

[4] https://puppet.com/

- Develop integration tests which explicitly test all workflows and service dependencies. Structure software so that mocks can be used in integration testing for all external dependencies.

- Carefully consider technology choices. While frameworks and 3rd party libraries can greatly reduce development time and cost, they also add risk. Avoid using trendy frameworks unless they provide real value to the project and can still be maintained if the original maintainers decide not to support them further.

- Make and publish static web archives (for example using a tool like webrecorder.io[5]) at significant milestones in development. This ensures that even if the software stops working, the scholarly work product it supported can still be accessed.

**How can curator/curating body help? (Actions, Q's, criteria, etc.)**
All of the actions outlined in the prior use cases A1 and A2 are relevant for this use case and the curator and/or curating body can provide additional guidance on several points:

- Offer guidance on how to evaluate the sustainability of services and and data sets on which the software depend. A useful approach might be to develop a set of criteria for researchers and developers to use in making these decision.
- Provide recommendations and advice on how to properly cite and credit external data sets and services.
- Identify the metadata standards and ontologies to be used for capturing internal and external software dependencies.
- Provide support for turning machine-actionable configuration manifests into descriptive accompanying user-facing documentation .

# Case B2

This use case deals with the curation of software that is part of a network of distributed services and linked data and supports an online digital publication of a scholarly work product. The software itself is not the primary research output but the research product is dependent upon it for its presentation. However, unlike use case B1, in this use case development is over and funding has ended.

**Salient qualities of use case**

- Research product is dependent upon it for presentation
- Part of a larger, service-based ecosystem
- Many external dependencies
- Development is finished and funding has ended

**Curation-readiness definition**
The software would be deemed to be curation ready if it can be distributed as a package that can be shared, whether for preservation and archiving, or reuse. It would contain documentation on installation/deployment, use and architecture/design; clear identification of any dependencies (with

[5] https://webrecorder.io/

dependencies included if possible), unit tests, and identification of all contributors to the development of the software, including individuals who participated in its design and testing as well as the actual coding.

**How to increase curation-readiness**

This use case has the same needs as B1 but if the recommendations in that use case were not taken into account and addressed prior to funding ending, curation options may be significantly limited.

The primary goal should be ensuring that the digital publication/scholarly work product that the software is responsible for presenting can continue to be accessed and reproduced.

Additional considerations:

- Use an application such as webrecorder.io to make an archive of the digital publication as it functioned at the end of the project funding.
- Preserve the pages of the publication as they appear at the end of project funding in the Internet Archive Wayback machine[6].

If possible, interview the developers and researchers to identify which aspects of the infrastructure supporting the digital publication may be most important to document and try to preserve in order to reproduce the research and its representation.

**How can curator/curating body help? (Actions, Q's, criteria, etc.)**

All of the points of prior use cases (A1-B1) are applicable to this use case. Additionally, help provide vocabulary and standards to use to describe static representations of the software and publication and identify repositories in which these artifacts may be preserved.

## Case C1

This use case is an example in providing outside researcher access to historic software from a collecting repository with a mission to preserve and present for posterity the artifacts and stories of the Information Age. The essential question concerning access is how much support can the institution provide a researcher? Just the bits or an emulated environment? Additionally, what level of effort do we engage in to guarantee the integrity of the historic software. Have we disk imaged it correctly or used another method of preservation that we can guarantee the authenticity of the software artifact over time?.

**Salient qualities of use case**

- **Legal implications** of access to licensed content where the collecting institution does not own the content. The majority of mass produced software is not owned by the person or institution who purchased it. The purchase only gave the purchaser a license to use the software. Therefore the repository does not have the right to freely provide access once the software artifact is donated to the repository. Can we make duplicate copies for preservation purposes?
- **Technical issues** either 'imaging' the original disk or providing access in original environment.
- **Access issues** around how much curatorial or archival support is provided to internal and external researchers and users.

[6] https://archive.org/web/

**Curation-readiness definition**
Curation ready defines the collecting repository's ability to legally and technically deliver historic software to a researcher. Along with metadata to support preservation and discoverability. It does not guarantee that the provided software will be accessible in a rendered environment but the repository should be able guarantee the authenticity of the code, that is that the bits have not been changed or altered from what was originally collected.

**How to increase curation-readiness**
The more resources we can provide up front for preservation and curation at the time of acquisition will greatly enhance our ability to provide access in the future. Technical actions such as 'imaging' the media and having a stable digital repository that is able to preserve the software are critical. Creating robust metadata records for description and discoverability will increase curation-readiness. Working to exploring licensing or other legal agreements that allow historical software (with no commercial viability) to be widely available online.

The above are the basics, taking it one step further we would provide an emulation environment along with enhanced disk image quality control to guarantee usability. As well as engaging in curatorial research to explore the reasons for the software's creation and previous use(s). In other words create a 'biography' of the software. This curatorial research may also include oral histories of the creator(s), videotaped demonstrations of working software, blog posts, or academic papers. All the questions under Case D2 concerning devising a Software Curation Profile are relevant.

**How can curator/curating body help? (Actions, Q's, criteria, etc.)**
Create best practices that can be followed by multiple institutions to streamline both preservation activities and make research availability quicker and easier. Best practices may include controlled vocabularies, providing basic emulation platforms or dependencies that can be shared online among institutions.

Work collaboratively on gaining licensing agreements with producers to grant access to non-commercially viable software for historic research. Provide access to printed materials such as manuals and user guides for historical software including allowing sharing of digitized copies between institutions and the the sharing robust metadata to enhance discovery and curation-ready materials.

# Case D1

A federal institution wanted to preserve and make accessible the software they developed from 1964 forward as a part of their institutional archives. Significant work was required to locate existent copies and compile contemporaneous documentation because software and software documentation was not included in record retention policies. Documentation was supplemented with oral histories with developers.

**Salient qualities of use case**

- Not under copyright

- Necessary in order to preserve institutional history
- Difficulties locating source code, executables, and documentation
- Curation takes place at the end of development

**Curation-readiness definition**
A software artifact, either executables or code, is deemed curation ready when it can be shared and executed on present-day devices and when metadata to support preservation and discoverability is created and stored along with the artifact. Documentation of the history of the artifact ought to also be created in order to assist in future exhibition or display.

**How to increase curation-readiness**
This use case has many similar needs as C1, but it is distinct because curation-readiness in this case means that the software can be executed on present-day devices. Because the majority of the labor for preservation in this use case was performed decades after the initial development, the most important characteristic that determines curation-readiness for a software artifact in this use case is that the artifact and related documentation can be located. Increasing curation-readiness, thus, requires ensuring the record retention policies explicitly include software artifacts and documentation related to the development, implementation, and use of said artifact.

Furthermore, where possible, it is desirable to move the preservation and/or curation workflows closer to the point of development. While it is possible to disk image and preserve some of the software artifacts located years after initial development, physical media does degrade past repair.

**How can curator/curating body help? (Actions, Q's, criteria, etc.)**
Although the use case is significantly different from A1 and A2, the actions listed in those use cases would further assist the curation-readiness of this use case, except the action related to licensing and copyright.

# Case D2

This use case considers legacy software collected as an institutional asset. In this case, both the software itself and related documentation (user manuals, technical reports) provide distinct informational value. Materials may be useful for a wide-range of scholarly endeavors, including: research on institutional histories (e.g., government-funded academic computing research programs), biographies (e.g., notable developers and/or contributors of software), socio-technical inquiries (e.g., extinct programming languages, implementation of novel algorithms), and educational endeavors (e.g., reconstruction of software).

**Salient qualities of use case**
- Hybrid collection of materials, spread across different domains and formats
- Software may or may not be accessible on magnetic media
- Paper printouts of source code, user manuals, requirements definitions, data dictionaries
- Wide range of possible uses and users for material

- Significant aspect of curation is assembling context for software

**Curation-readiness definition**
Assembled software and related documentation that is ready to be discovered, interpreted, cited, and made accessible. In this use case, curation of materials most likely begins after creation. Thus, increasing curation-readiness is functionally equivalent to the actions that a curating body might take on a set of materials.

**How to increase curation-readiness**
*Identify and assemble relevant materials.* A significant challenge with this use case lies in the assembling of relevant materials to provide necessary context for meaningful access and use of materials. Inventorying potential materials of interest that have some relationship to the software may be a useful starting point. Established appraisal criteria can be used to guide decisions about selection of materials for access and long-term retention. Finally, identifying custodians and stakeholders, either at or outside of the institution, will help ensure proper transfer of materials and rights management issues, where applicable.

*Describe and catalog materials.* Curation-readiness can be increased by thoroughly describing and cataloging selected materials, including relationships between documents. Although the software itself may not be accessible, providing access points through description to the accompanying materials (i.e., printouts of source code, technical requirements documentation) may provide ample and sufficient context for use. A potential area ripe for further exploration is exploring different access scenarios for users of hybrid collections.

*Digitize and OCR paper materials.* Paper printouts of source code and related documentation can be digitized according to established best practice workflows. The use of optical character recognition (OCR) programs produces machine-readable output, enabling indexing of content for discoverability and/or textual transcriptions. The latter option can make historical source code more portable for use in simulations or reconstructions of software..

*Migrate media.* Legacy software may reside on unstable media. In cases where access to the software itself is desirable, migrating and/or extracting media contents (where possible) to a more stable medium is recommended.

**How can curator/curating body help? (Actions, Q's, criteria, etc.)**
A recommended curatorial strategy is to devise a Software Curation Profile for a set of materials, prompting the capture of valuable contextual information through documentation.
The following questions provide some guidance for creating Profiles:

- *Who created the software? For what purpose and function?*
- *Who used the software? For what purpose and function?*
- *What user documentation accompanied the software? (e.g., user manuals)*
- *How was the software intended to be used?*

- *What programming language is the software written in? What are the dependencies? What was the original computing environment?*
- *Where did materials originate? How were they assembled?*

# Discussion and Comparison

In order to compare across use cases, Table 1 shows a generalized set of preservation requirements extracted from the use cases themselves. The importance of each requirement is then listed for each use case. Table 2 shows a generalized set of curation-readiness actions along with the importance of that action to each use case. The importances shown in Tables 1 and 2 represent a subjective and idealized but grounded view of what we (the authors) consider important for each use case.

Although there are significant differences in some of the details of research vs non-research (i.e., museum/archival) use cases (A1 to B2 vs everything else), we can see from Table 1 that the most important characteristics that should be considered for software preservation across use cases are:

- The need for documentation. This is highly important as it gives specific details of the software in the particular use-case. These details are (in part) what allows the understanding and re-use/re-execution of the software
- Licensing issues. Identifying the terms under which software can be made available and actually attaching those terms explicitly to software are deemed highly important for the large majority of cases.
- The need for crediting contributors is deemed highly important for the large majority of use cases.
- The need for a ready-to-build or ready-to-run package. In combination with documentation, having a package that can easily be run or built at least on one computer system would make the package more easily preservable (subject to the specifics of the use case).

The above findings are important but unsurprising. The first two bullets confirm what has been previously stated in the literature. The third bullet point has not often been emphasized as an important part of software preservation but without established practice of crediting those who contributed to the creation of software (and it's preservation), there is less incentive to adopt curation-ready practices. The final bullet point represents a common-sense result that is difficult to attain in practice and as a result of the particularities of each use case, general statements on how to achieve it do little to help the situation. However, by analyzing the findings in the context of the research vs non-research use cases, we can gain a deeper insight into how the ready-to-build or ready-to-run goal can be addressed for classes of use cases.

In the research use cases, following development best-practices where possible is deemed highly important. This is because these practices are largely a result of established (open source) software engineering practice which, on paper at least, address many of the issues that make preservation difficult (e.g., lack of documentation, source code management, testing, use of difficult or costly to obtain dependencies, lack of automation and configuration management for robust deployment, etc). These practices are obviously not applicable to the cases where software development has ceased.

**Table 1: Importance of preservation requirements across use cases. Cases A1 to B2 are from the research perspective. The remaining cases involve software from the museum/archival perspective.**

● = highly relevant/important ◐ = somewhat relevant/important ○ = little relevance/importance blank = not relevant or not applicable

| Reqs. > | Follow development best practices | Document | Manage Dependencies | | Package the software to enable redistribution & re-execution | | | |
|---|---|---|---|---|---|---|---|---|
| Use Case v | Version control, documentation; portability; unit & integration tests; non-proprietary platforms & libraries | Purpose, usage, APIs, dependen-cies | Explicitly version and/or track | Keep local copies | Attach licenses (once identified) | Use virtual machines, containers, emulators | Create ready to build / run package | Use configuration management and automated deployment methods |
| A1 | | ● | ● | ◐ | ● | ○ | ● | ○ |
| A2 | ● | ● | ● | ◐ | ● | ○ | ● | ◐ |
| B1 | ● | ● | ● | ◐ | ● | ● | ● | ● |
| B2 | ● | ● | ● | ◐ | ● | ● | ● | ● |
| C1 | | ● | | ● | ● | ◐ | ● | ○ |
| D1 | | ● | | | | ● | ◐ | |
| D2 | | ● | | | | ◐ | ◐ | |

**Table 1 (cont'd.)**

| Reqs. > | Make and publish static archives | Credit all contributors | Collect & package ancillary documentation | Evaluate sustainability policies and long-term viability | Legal / Contractual | | |
|---|---|---|---|---|---|---|---|
| Use Case v | Web page, screenshots, video capture | | Printed material, associated publications | | Handle archiving of commercial code, copy protection bypass, etc. | Identify the terms under which software can / should be made available | Secure re-distribution rights |
| A1 | | ● | ○ | ◐ | ◐ | ● | |
| A2 | | ● | ○ | ● | ◐ | ● | |
| B1 | ◐ | ● | ○ | ● | ● | ● | ● |
| B2 | ● | ● | ○ | ◐ | ● | ● | ● |
| C1 | ● | ● | | ● | ● | ● | ● |
| D1 | | ◐ | | | | ◐ | ○ |
| D2 | ◐ | ◐ | ● | ◐ | ◐ | ◐ | ◐ |

**Table 2: Kinds of actions that can be taken, questions asked, or criteria met to address the requirements from Table 1. Cases A1 to B2 are from the research perspective. The remaining cases involve software from the museum/archival perspective.**

● = highly relevant/important ◐ = somewhat relevant/important ○ = little relevance/importance blank = not relevant or not applicable

| Question category > | Clarify preservation objectives and audience | Identify the purpose of the software | Assist with license selection | Examine software granularity | Recommend development best practices | Identify tools and ontologies for dependency tracking | Apply workflow capture tools | Identify citation & credit best practices |
|---|---|---|---|---|---|---|---|---|
| Use Case v | | | | | | | | |
| A1 | ● | ● | ● | ○ | | ○ | ◐ | ● |
| A2 | ● | ● | ● | ○ | ● | ◐ | ○ | ● |
| B1 | ● | ● | ● | ● | ● | ● | ● | ● |
| B2 | ● | ● | ● | ● | ● | ● | ● | ● |
| C1 | ● | ● | ● | ◐ | ○ | ○ | ● | ● |
| D1 | | ◐ | | | | | | ● |
| D2 | ● | ● | ◐ | ◐ | | ◐ | | ● |

In contrast, the use of virtual machines, containers, or emulators is applicable to all cases. Nevertheless, there were differences between the cases on how important these technologies are to achieve the preservation goals. For cases where development had ceased or for highly complex research software, packaging up software into a (nearly) ready-to-run package is deemed highly important whereas it was not as important for simpler cases or cases where old hardware/software environments are still available.

Dependency management showed some interesting commonalities and differences between use cases. From Table 1, explicitly versioning and/or tracking dependencies is critical for the research-oriented cases, as one would expect, due to the intensive software development work involved. Since software development usually has ceased, the archival perspective cases do not emphasize tracking dependencies. Keeping local copies of dependencies is deemed somewhat important for the research cases. This is because during development, dependencies for compiling the software are more readily available compared to some time after development has ended. The museum/archival cases deemed this facet not applicable due to the assumption that software (especially commercial software) generally comes with the majority of the difficult-to-obtain dependencies. C1 is the exception due to the increased focus on software that can actually be executed vs the other museum/archival cases where execution is not as critical. In Table 2, even the cases that deemed dependency tracking as highly relevant in Table 1 do not place a large importance on tools or ontologies to do so. This suggests that having formal tools to track dependencies is less important that simply tracking them in *some* way.

From a planning perspective, Table 2 suggests that having clarity on what the software does and what the preservation objectives are, is one of the most important parts of increasing the curation-readiness of software. Due to the particularities of each use case, having clear answers to these questions will make it easier to identify what aspects of the software need the most attention and what methods/tools are available that might enable its preservation.

# Final words and Future Work

When it comes to preservation, the most important aspects to consider, regardless of use case, are documentation, licensing/intellectual property, credit, and having ready-to-use software. Although these aspects apply to basically every case presented, the specifics of how each requirement is addressed can vary. For example, licensing issues for research software may involve determining an appropriate open source license under which to distribute the software. On the other hand, from the museum perspective, just obtaining permission to distribute code may be the most challenging aspect. Furthermore, the meaning of having “ready-to-use” software could vary greatly in difficulty depending on the case.

Although this report presents actions to take for preserving software in the context of specific cases, due to the relative newness and inherent abstractness of preserving software (as opposed to physical objects), solutions to many of the difficult details are not given simply because they have not yet been developed. It is the intent that the groundwork laid in this report can be built upon by those tasked with preserving software so that some of the difficulties may be addressed and best practice solutions developed. One example of how solutions could be developed comes from the research software arena. In this area, one potential way to quickly trial curation-ready software development practices is to incorporate some of the proposed actions into hackathon environments. For example, having a template that asks developers quick questions at key points would increase the ease with which the hackathon output could be curated for better understanding and sharing. E.g.,

- At the start of the hackathon, ask about documentation plans, tool adoption for dependency management and version control, and purpose and audience of the software.
- At the end, ask about what was accomplished, where can it be found, and who contributed.

Because it’s impossible for the use cases in this report to cover all possible software preservation scenarios, thinking through and attempting to identify the kind of information presented in Rios et al. (2017) and in this report is, we believe, a useful starting point for determining how the particularities of other use cases might be tackled. It follows that verifying whether the actions for increasing curation readiness presented in this report were actually worth it (i.e., resulted in easier preservation relative to cost) would be a useful endeavor. Therefore, identifying partners that can apply (and refine) these actions to other use cases is the next logical step in moving towards making software more curation-ready.