Guidelines for Managing and Sharing Research Artefacts in SE

In this document, we report a list of practices for managing and sharing research artifacts extracted from a set of guidelines that have been recommended by major computer science publishers, venues, and organizations. The set of guidelines analyzed is composed by the following documents:

1. The ACM Artifact Review and Badging [1]
2. The EMSE OpenScience initiative [2], [3]
3. The Journal of Open Science Software (JOSS) [4]
4. The Journal of Open Research Software (JORS) [5]
5. The Guidelines by Wilson et al. [6]
6. The NASA Open Source Software Projects [7]
7. The TACAS artifact evaluation guidelines [8]
8. The CAV artifact evaluation guidelines [9]

# The ACM Artifact Review and Badging

Below we report a list of practices and properties extracted from [1]

* Software system
* Scripts to run experiments
* Input data
* Raw data collected
* Scripts to analyze results
* Sufficient description
* Measurement procedures
* Measuring system
* Operating conditions
* Instructions on how to obtain proprietary/third-party artifacts
* Documentation to facilitate reuse/repurposing
* Norms and standards of the research community that were adhered
* Permanently available for retrieval
* Publically accessible archival repository
* Unique identifier (e.g., DOI, URL)
* Plan to enable permanent accessibility
* Preferably they should not be published on personal web pages
* Relevance to the study
* Adds value beyond the text in the article
* Data from which the figures were drawn
* Implementations of algorithms
* (Complete?) software under study
* Experimental protocol
* Independent replication and reproduction
* Formal audition
* Verify robustness
* Build upon previous work through reuse and repurposing
* Enable artifacts to be exercised
* Contribute
* Well-formed and documented artifacts
* Documented, Consistent, Complete, Exercisable
* Multiple trials

# The EMSE OpenScience initiative

Below we report a list of practices and properties extracted from [2]

* Increase transparency, reproducibility, and replicability
* Faster scientific progress
* Build on the results of older studies
* Raw data
* Material necessary for the analysis and interpretation of raw data
* Study protocols
* Analysis scripts
* Sharing (disclose) data sets
* Replication packages
* Minimal disruption
* review guidelines
* open data sets

Below we report a list of practices and properties extracted from [3]

* open data
* open code
* anonymized and curated empirical data
* source code
* replication package
* increase transparency and replicability
* support the immedite credibility of author's work
* common basis for joint community efforts grounded on shared data
* increase transparency
* "reproducible, fully understood, replicated"
* data-sharing culture
* publicly archive their data and related material
* required to understand and reproduce the claims
* can't disclosure data? Provide an explicit and short statement
* remain available in the very long term (e.g., zenodo, figshare)
* use an appropriate license (e.g., CC-BY 4.0, MIT)
* Replication package
* FAIR -> Findable, Accessible, Interoperable, Reusable
* Avoid putting data in their own (institutional/private) websites or systems (e.g., dropbox, VCS) or systems (e.g., academia.edu, researchgate)
* Personal websites are prone to changes and errors. They are 30\% likely to stop working in a 4 years period
* Package disclosured should link to the paper (DOI) upon final production
* Allow updating published replication packages any time
* Allow others to understand the claims and analyses
* Instrumentation
* Transcripts (e.g., from interviews - potentially anonymized)
* field notes
* codebooks
* state reasons for confidentiality of research dataa or artefacts
* UML diagrams/models
* Requirements text
* design documents
* machine-readable and open formats (e.g., JSON, XMI -- as opposed to Visio, PNG, PDF)
* Downloadable code and data
* Single URL
* Hosted on a persistent, archived repository
* Inventory of artifacts (e.g., files and folders)
* used file formats documented
* naming conventions
* complete (i.e., contains everything required to understand and/or re-compute data, numbers and figures) in the paper
* package includes instructions to compile and execute the code
* dependency on publicly available modules/libraries
* appropriate license for code or data

Below we report a list of practices and properties extracted from [10].

* Associated data, code, or both required to study and reproduce the results of a published paper
* To fully understand the contribution
* To verify the claims
* To build upon the results
* Replication package, Laboratory package, supplementary material, Online appendix
* Downloadable: data and code lies behind a single URL (e.g., DOI)
* Findable: Well-indexed repository (e.g., findable by search engines based on the paper title)
* Documented: inventory of artifacts (files and folders),
* used file formats
* naming conventions
* Complete: numbers (i.e., data) and figures can be re-computed
* Exercisable: code must be executable, does not depend on non-publicly available modules and libraries
* Durable: Repository URL must be stable in the long term (e.g., 10-100 years? More? – See GitHub Arctic Code Vault project <https://archiveprogram.github.com/>)
  + Zenodo by CERN
  + FigShare
  + HAL by CNRS
  + archive.org by the Internet Archive foundation
  + OSF.io by the Center for Open Science
  + GitHub
* Versioning
* Replication
* Communication (issue tracker)
* Github (as working repo) + Zenodo (backup archive) = Stable, DOI
* Completeness more important than Size
* Raw data then Intermediate data then Final data then Table, Graphics, and numbers (i.e., Sequence of steps to facilitate the use by future researchers)
* Intermediate steps may help to understand and fix problems
* Licenses (e.g., that maximize impact, back-contributions and control)
* Other points:
  + main file that is easily identifiable, e.g., Readme file \cite{prana\_categorizing\_2019}
  + all file formats documented (e.g., meaning of rows and columns)
  + restoration procedure documented (e.g., for database dumps)
  + used versions of artifacts
  + compilation and execution processes documented
  + for studies involving humans, training material should be made available and documented
  + contact points documented (e.g., email, issue tracker)
  + README tell which paper(s) to cite
* Container images, such as docker, [...] for freezing dependencies
* virtualization images [...] to get a running environment quickly
* Dependency management systems (MAVEN, PIP) [...] to specify dependencies
* Jupyter Notebooks [...] to document and typeset research code and results
* Open-science [...] for outreaching to society
* Beautiful repository with nice illustration
* Illustration of scientific data and code [...] for citizens
* Appeal to wider audience than our scientific peers

# The Journal of Open Science Software (JOSS)

Below we report a list of practices and properties extracted from [4].

* 90\% of academics use software in research
* more than 70\% say that their research would be impractical without software
* publication, acknowledgement and citation
* research article and grants are the main indicators of research productivity
* paper about the software (i.e., focused on the software itself)
* licensed code
* has good documentation
* built on github (i.e., issue tracker, rapid interaction)
* open access
* well-documented
* public repository
* short paper in markdown format to declare authors, purpose of the software, any needed references
* indexed
* citable
* markdown \texttt{paper.md} visibly located
* source code available
* OSI-approved license
* software versions in the paper vs. repository matching
* performance claims
* documentation
* statement of need for the software
* installation instructions
* example usage
* documentation of functionality
* tests
* community guidelines
* author list
* well-structured references
* permanent archive
* DOI

# The Journal of Open Research Software (JORS)

Below we report a list of practices and properties extracted from [5].

* Software downloadable based on the information provided
* The software behaves as described
* Accuracy and quality of the metadata (e.g., list of contributors, license, limitations)
* indication of the software's functionality
* where it would be used
* keywords enable [...] to search for the software
* Introduction: Enough background information to understand the context of the software development's and use
* Implementation and Architecture: How the software is designed, and constraints that may be placed on its use
* Quality control: How results can be trusted
* Reuse: suggestions for other potential applications, ways of extending or modifying the software, integration with other software
* Figures and Diagrams
* Suitable repository
* suitable open license
* Persistent identifier (i.e., DOI)
* Link to download the code
* Sample input/output
* code adequately documented (e.g., in-code citation \cite{alsudais\_incodecitation\_2021})
* how to build/deploy/install/run the software
* how to identify if the software is operating as expected (e.g. smoke test)
* support mechanism
* provide info useful for the comminity
* clear to read
* defines unfamiliar concepts
* recognize efforts in the area
* credibility and trustworthy
* summarises the experiences of the authors
* authors' depth of understanding
* ORCID
* all data openly available
* FAIR principles
* Data Accessibility Statement
* data obtained from other sources appropriately credited
* repository suitable for the subject and has a sustainability model
* deposited under open license (explicit)
* include a version that is in an open, non-proprietary format
* data is labeled (e.g., column headers, descriptiuon in readme text file)
* Anonymized data
* funding sources
* detail requirements for ethical research

# The Guidelines by Wilson et al.

Below we report a list of practices and properties extracted from [6].

* Basic lab skills
* Data can get lost
* Analyses can take much longer
* how effectively researchers can work with software and data
* organized data, documented steps project structured for reproducibility

1. Data management:
   1. Save the raw data
      1. or record the exact procedure used to obtain the raw data
      2. as well as any other pertinent information
   2. Ensure that raw data are backed up in more than one location.
      1. e.g., External HDs, university-owned data storage, incremental/special backup
   3. Create the data you wish to see in the world.
      1. machine and human readability
      2. Use open/non-proprietary file formats, self-explaining variable names/data codes
      3. Useful meta-data as part of filenames (e.g., for pattern matching)
      4. Follow research community standards for how to provide meta-data
      5. Metadata about: the dataset as a whole AND the content content within the dataset
   4. Create analysis-friendly data.
      1. e.g., each column is a variable, each row is a observation
   5. Record all the steps used to process data.
      1. Write scripts for every stage of data processing (e.g., to redo data preparation, collection)
      2. Produce intermediate data files, with increasing levels of cleanliness and task specificity
      3. Breakdown workflows is better than designing \textit{monolithic procedure} (i.e., prefer the explicit creation/retention of intermediate products)
   6. Anticipate the need to use multiple tables, and use a unique identifier for every record.
   7. Submit data to a reputable DOI-issuing repository so that others can access and cite it.
      1. e.g., Figshare, Dryad, Zenodo
2. Software
   1. Place a brief explanatory comment at the start of every program.
      1. one example of how to use
      2. reasonable values for parameters\
      3. help menu
   2. Decompose programs into functions.
      1. about 60 LOC
      2. no more than 5-6 parameters
      3. low coupling and high cohesion
   3. Be ruthless about eliminating duplication.
      1. avoid copy-paste
      2. use data structures
   4. Always search for well-maintained software libraries that do what you need.
   5. Test libraries before relying on them.
   6. Give functions and variables meaningful names.
      1. language conventions (e.g., \texttt{CamelCase}, \texttt{snake\\_case})
      2. Tab completion
   7. Make dependencies and requirements explicit.
      1. e.g., requirements.txt, \textit{Getting started} in the README file
   8. Do not comment and uncomment sections of code to control a program's behavior.
      1. use if/else structures, parameters
   9. Provide a simple example or test data set.
      1. build-and-smoke test
   10. Submit code to a reputable DOI-issuing repository.
       1. e.g., Zenodo integrates with GitHub
3. Collaboration
   1. Create an overview of your project.
      1. e.g., purpose, project title, description, contact info, running example
      2. setup local workspace
      3. find tasks to contribute (CONTRIBUTING file)
      4. dependencies to be installed
      5. tests to ensure proper installation
      6. guidelines/checklists that your project adheres to
   2. Create a shared "to-do" list for the project.
      1. Make it easy for newcomers (e.g., grad students, postdocs, future-you)
      2. create shared to-do list (\texttt{notes.txt} or \texttt{todo.txt} reporting issues/limitations)
   3. Decide on communication strategies.
      1. communication channels
      2. e.g., mailing lists, chat channels, documentation, meeting notes
   4. Make the license explicit.
      1. make license explicit (LICENSE file)
      2. ease giving you credit
   5. Make the project citable.
      1. CITATION file to indicate how to cite
      2. where to find datasets/code/figures/other artifacts
4. Project organization
   1. Put each project in its own directory, which is named after the project
      1. divide work based on the overlap in data and code files
      2. organize project by the types of files to effectively find and use things later
      3. consistency and predictability vs. "hairsplitting"
   2. Put text documents associated with the project in the doc directory
      1. \begin{itemize}
      2. manuscripts
      3. documentation for source code
      4. electronic lab notebooks
      5. subdirectories for different classes
      6. \end{itemize}
   3. Put raw data and metadata in a \texttt{data} directory and files generated during cleanup and analysis in a \texttt{results} directory
      1. raw data and metadata - subdirectories for time, method of collection or other metadata
      2. cleaned data, statistical tables, publication-ready figures or tables
   4. Put project source code in the \texttt{src} directory
      1. Distinguish files by clear filenames or directory
      2. e.g., programs in interpreted languages, shell scripts, snippets
      3. core analysis (i.e., \textit{scientific gut})
      4. controller/driver scripts (e.g., analysis steps, particular parameters, data IO commands)
   5. Put compiled programs in the \texttt{bin} directory
      1. i.e., executable programs
      2. There is bin, if these files exist
   6. Name all files to reflect their content or function
5. Tracking changes
   1. Back up (almost) everything created by a human being as soon as it is created
   2. Keep changes small
      1. edits that you could imagine wanting to undo
   3. Share changes frequently
   4. Create, maintain, and use a checklist for saving and sharing changes to the project
      1. log messages explaining changes (e.g., CHANGELOG.txt, commit messages)
   5. Store each project in a folder that is mirrored off the researcher's working machine
      1. remote version control system
   6. Add a file called \texttt{CHANGELOG.txt} to the project's docs subfolder.
   7. Copy the entire project whenever a significant change has been made.
   8. Use version control system
      1. enable to reference or retrieve specific version aid to reproducibility
6. Manuscript
   1. Write manuscripts using online tools
      1. rich formatting
      2. change tracking
      3. reference management
   2. Write the manuscript in a plain text format
   3. it permits version control
7. Supplementary materials
   1. branches
   2. build tools
   3. unit testing
   4. coverage
   5. continuous integration
   6. profiling and performance tuning
   7. semantics web
   8. documentation
   9. bibliography management (e.g., Zotero, ORCID)
   10. code review, pair programming

# The NASA Open Source Software Projects

Below we report a list of practices and properties extracted from [7].

* Project meta-data
* Good tags for your code project
* description of the project to be released
* individuals involved in its creation
* development timeline
* available documentation
* related topics
* provide details about external dependencies (e.g., packages or sources)
* terms and conditions (e.g., license)
* 508 compliance: accessible to, and usable by, individuals with disabilities.
* software engineering practices

# The TACAS artifact evaluation guidelines

Below we report a list of practices and properties extracted from [8].

* substantiates the claims made in the paper
* ideally makes the claims fully replicable
* tool (e.g., in binary or source code format)
* documentation
* input files (e.g., models analysed or programs verified)
* configuration file or parameters used in the experiment
* easy-to-use scripts (e.g., run experiment and draw tables/graphs)
* abstract that summarizes the artifacts and its relation to the paper
* LICENSE file
* README file that contains detailed step-by-step instructions on how to use to repliace
* SHA-256 hash
* virtual machine
* how-to file
* additional software or libraries shall be included
* instructions must include all necessary steps for their installation and setup
* work without a network connection
* how to replicate most, or ideally all, tof the experimental results
* detailed documentation assuming minimal expertise of users
* provide a way to replicate a subset of the results with reasonably modest resources
* state resource requirements or the environment in which you successfully tested the artifact

# The CAV artifact evaluation guidelines

Below we report a list of practices and properties extracted from [9].

* final PDF
* instructions on how to run the tool
* virtual machine
* compute SHA1 checksum
* short plain-text file describing the OS and parameters of the image and host platform
* how to proceed after booting the image
* instructions for locating the full documentation for evaluating the artifact
* Include accepted paper
* detailed README of how to run the tool
* directory containing the benchmarks + tool or proof scripts
* easy to evaluate and yields expected results
* how to reproduce most of the experimental resilts
* keep the process simple
* detailed documentation assuming minimum expertise
* should work without a network connection
* reasonably modest resources
* include simpler benchmark
* point the most relevant and interesting parts of the source tree code
* include log filesthat were produced by their tools
* point to relevant log files in the artifact description

# References

[1] ACM, ‘Artifact Review and Badging - Current’, Aug. 24, 2020. https://www.acm.org/publications/policies/artifact-review-and-badging-current (accessed Mar. 18, 2021).

[2] D. Méndez Fernández, M. Monperrus, R. Feldt, and T. Zimmermann, ‘The open science initiative of the Empirical Software Engineering journal’, *Empir Software Eng*, vol. 24, no. 3, pp. 1057–1060, Jun. 2019, doi: 10.1007/s10664-019-09712-x.

[3] EMSE, ‘EMSE Open Science Initiative’, Mar. 18, 2021. https://github.com/emsejournal/openscience/blob/master/README.md (accessed Mar. 18, 2021).

[4] D. S. Katz, K. E. Niemeyer, and A. M. Smith, ‘Publish your software: Introducing the Journal of Open Source Software (JOSS)’, *Computing in Science Engineering*, vol. 20, no. 3, pp. 84–88, May 2018, doi: 10.1109/MCSE.2018.03221930.

[5] JORS, ‘The Journal of Open Research Software - Editorial Policies’, Feb. 17, 2021. http://openresearchsoftware.metajnl.com/about/editorialpolicies/ (accessed Mar. 18, 2021).

[6] G. Wilson, J. Bryan, K. Cranston, J. Kitzes, L. Nederbragt, and T. K. Teal, ‘Good enough practices in scientific computing’, *PLOS Computational Biology*, vol. 13, no. 6, p. e1005510, Jun. 2017, doi: 10.1371/journal.pcbi.1005510.

[7] NASA, ‘NASA Open Source Software’, 2021. https://code.nasa.gov/ (accessed Mar. 18, 2021).

[8] TACAS, ‘TACAS 2019 - ETAPS 2019’, 2019. https://conf.researchr.org/track/etaps-2019/tacas-2019-papers#Artifact-Evaluation (accessed Mar. 18, 2021).

[9] CAV, ‘Artifacts | CAV 2019’, 2019. http://i-cav.org/2019/artifacts/ (accessed Mar. 18, 2021).

[10] M. Monperrus, ‘How to make a good open-science repository?’, *Research Data at Springer Nature*, Dec. 11, 2019. https://researchdata.springernature.com/posts/57389-how-to-make-a-good-open-science-repository (accessed Mar. 19, 2021).