Why do we need to compare research software, and how should we do it?

Why Do We Need To Compare Research Software, And How Should We Do It?

Neil P. Chue Hong
Software Sustainability Institute
University of Edinburgh
JCMB, Peter Guthrie Tait Road
Edinburgh, EH9 3FD, United Kingdom
N.ChueHong@software.ac.uk
ORCID: 0000-0002-8876-7606

Abstract— **How can we measure research software? Is it possible to compare it, given the myriad different research domains, practices and pieces of software? Do we even need to do this, and what benefits might it bring? This position paper sets out the reasons for why different stakeholders, from users to developers to funders, might wish to undertake this difficult task, and describes a proposed framework for doing so (based around measures of accessibility, usability, maintainability and portability) which takes into account the possibility of variation between different communities about how they prioritise different aspects of research software.**

Index Terms **— software, research software, software metrics, software engineering, software sustainability.**

I. Background

Software underpins much of the scientific research undertaken today. As well as the "traditional" use of software for modelling and simulation, it is used to manage and control instruments, and analyse and visualise data. A challenge for users, funders, and developers of scientific software is how to determine how *"good"* a piece of software is – in large part because they have differing notions of what *"good"* means. This has implications for the software's usability and reusability, as well as its impact and return on investment.

An incredible amount of investment of effort and money is put into scientific software. In the UK, the Engineering and Physical Sciences Research Council (EPSRC) estimated that it had invested approximately £9m per annum on scientific software [1]. Studies done by the Software Sustainability Institute suggest that a total of £840m was invested in RCUK research grants that rely on software to undertake the proposed research [2]. It is clear that something is required to make it easier to understand which software should be promoted, which software reused, and which software retired. The question is what this should be.

In the UK, the e-Infrastructure Leadership Council[1] created a Software Taskforce to identify issues related to scientific software, and make recommendation for how the impact of scientific software could be improved. As part of this, the concept of a Software Accreditation Framework was discussed. This initial concept was similar to the "traffic light" labelling scheme for nutritional information applied in the UK [3], where information on each of a defined set of categories are displayed on packaging based on both standardised and serving based sizes, highlighting levels using red/amber/green colouring to give guidance to consumers.

One of the benefits of such a system is that it gives a small set of quantitative indicators for a consumer to look at, and the colouring can be used in different ways depending on the type of product. For instance, whilst a cake and a salad will have very different measurements for fat, it is not the case that a high (red) fat level is bad for both. Many people would choose a cake with a high fat level because they prioritised other factors associated with fat (e.g. as a proxy for taste).

Similarly for research software, there are differing priorities which might factor into a users choice. Rapid releases might be welcomed as a sign of continued development and support by some users, whereas for others it is an indication that the software changes too fast to be integrated into a larger stable system. Likewise some users may prioritise extensibility (e.g. to use a different chemical structure model) whilst others may prioritise performance – it would be impossible to come up with a single metric that satisfied both, hence the need for a software assessment framework that could be used to provide ways for different stakeholders to make informed and appropriate decisions, based on their own requirements.

The benefits of assessing software in this way might include:

- Providing measures understandable to an end-user which are reasonable proxies for software quality
- Facilitating discovery and choice of software by other researchers and small and medium-sized enterprises (SMEs)
- Improving software reuse and commercialisation
- Increasing recognition for developers
- Increasing recognition for good software development practice
- Creating a basis for a marketplace for research software

[1] https://www.gov.uk/government/groups/e-infrastructure-leadership-council

To ensure adoption and impact, it is important that the use of this framework is both bottom-up (encouraging code owners to be proactive in getting their codes accredited); easy-to-use (with objective measures); simple (to avoid confusion); enable community norms (be understanding of the relative priorities of different communities); and minimise game playing.

This means that wherever possible, a framework should concentrate on objective *measures*, which can be used to guide subjective *indicators*.

II. Related Work

Many aspects of what one might choose to measure about software have been investigated separately.

Models have been proposed to define such elements as cost models for software reuse [4][5], maturity models [6][7], and component reusability [8][9] across software in general. This author's own Five Stars of Research Software [10] posited a set of categories which "good" scientific software could be assessed on, but did not fully define how each category could be objectively measured.

The software engineering community has developed a large body of knowledge regarding software quality and the measurement of software [11][12] and more recently looking at the various aspects of sustainability beyond natural/environmental (green) sustainability [13]. However much of this knowledge is not widely disseminated within the research software community, either because it is not openly licensed, or it is perceived to only be relevant to large commercial software projects with large resources.

Certain aspects of the software source code can be defined and automatically measured, for instance McCabe's (cyclomatic) complexity, code metrics (including source lines of code (SLOC), number of comment lines), and Halstead's complexity metrics (based on numbers of operators and operands) all of which look at measures of size and complexity which may make a program harder to understand, more difficult to maintain, and increase the likelihood of errors. Visual Studio, a common development tool, calculates its own Maintainability Index [14] that represents the relative ease of maintaining the code based on a combination of these measures and this has been implemented in other tools for popular languages like Python (Radon) and JavaScript (JSComplexity). However a significant issue with these types of metrics is that they are based on work from the 1990s using data from very different types of programming languages and software from modern systems, which means they may not have relevance for contemporary codes [15].

Other work focuses specifically on research software. Depsy[2] is a tool which mines papers to find fulltext mentions of software (currently only those written in Python and R) and analyses GitHub repositories to see where software is being used, thus giving a measure of the impact of a piece of research software or library. Ontosoft [16] defines an ontology for software which categorises many of the aspects that intuitively are part of understanding its quality. The Software Discovery Dashboard[3] aims to search multiple code hosting services, such as Zenodo, Figshare, and GitHub, for scientific software and undertake analysis of it, utilising the Codemeta[4] metadata standards for describing scientific software and building on previous work by Mozilla Science Lab, Github and Figshare on Code as a Research Object[5].

Another set of work looks at certifying the process of production. For instance, the Data Archiving and Networked Services (DANS) Data Seal of Approval[6] enables self-certification and peer review of data repositories to ensure they are storing research data in a reliable manner such that the data can be accessed and reused, and are seeking to create a Software Seal of Approval, focussed on software projects. The Open Data Institute's Open Data Certificates[7] uses a similar questionnaire-style assessment to recognise open data that has been published in a sustainable and reusable way. The Software Sustainability Institute's online sustainability evaluation tool[8] enables developers to self-evaluate the reusability and maintainability of their research software.

Finally, there is a body of work looking at the indicators of success for open source projects, which in some ways are similar to research software projects due to the nature of the teams, but differ because of their differing goals. Research has examined projects in major public repositories to identify potential indicators [17][18] and categorise different levels of success in initiating and growing projects [18][20]. The identification of time-invariant vs time-variant variables and the application of machine learning techniques has been used to identify potential indicators for open source project success [21].

III. Defining A Framework

As mentioned above, the success of nutritional information comparison has relied on defining a relatively small number of "headline measures". For research software, we wish to define a similar small set of measures which enable the key characteristics of:

- Availability: ability for the software to be found and obtained by a potential user
- Usability: ability for the software to be used, integrated and extended by a potential user
- Maintainability: ability for the software to be sustained by developers
- Portability: ability for the software to be used in areas outside its original user base, for instance subject area, operating system, hardware architecture or different type of user.

These terms are used more loosely than the definitions commonly used in software engineering, to reflect the differing perceptions of software by typical research users.

[2] Depsy: http://depsy.org/

[3] Software Discovery Dashboard: https://github.com/mozillascience/software-discovery-dashboard

[4] CodeMeta: https://github.com/codemeta/codemeta

[5] Code as a Research Object: https://science.mozilla.org/projects/codemeta

[6] Data Seal of Approval: http://www.datasealofapproval.org/en/

[7] Open Data Certificates: https://certificates.theodi.org/

[8] Online sustainability evaluation: http://www.software.ac.uk/online-sustainability-evaluation

In defining these characteristics, the author has drawn on their experience, and that of their colleagues in working in over 100 research software projects of different sizes and in different domains. This paper documents a synthesis of previous ideas, aiming to narrow it down to a useful subset of all potential measures of research software in an attempt to create a set which can be used for further study and checking, by assessing existing software in a follow-up study.

A. Availablity

There are two key parts to availability: can a user find the software (discovery), and can they obtain the software (access)?

The important metadata associated with this category include:

- Discovery and Choice
  - Software Name: as this is often used as a search term
  - Software Description: which can be matched in searches and read by users
  - Categories: such as operating system support, type of software
  - Keywords: again used for search terms
  - Features
- Access
  - Website: as this is often where scientific software is distributed
  - Repository: for source code and possibly binary distribution
  - License: governing the terms of use and reuse

The proposed measures for this category are:

- Is the software easy to find, if a user attempted to search for it based on name or keywords?
- Can the software be downloaded from the website or repository addresses provided?
- Is the license obvious, and is it commercial, copyleft, permissive, academic, or public domain?
- Is the software widely used by other members the users' community? Does it have a high market share of the potential market (which may be small for niche products)?

B. Usability

This category takes into account the ability of a user or developer to obtain information that enables them to understand the operation of the software, such that they can use it, integrate it with other software, and extend or modify it.

The important metadata associated with this category include:

- Execution
  - Documentation / Installation Instructions
  - Operating System
  - Dependencies
  - Development Status
  - Performance benchmarks
- Integration
  - Input data formats
  - Output data formats
  - Programming Language
- Extension
  - API documentation
  - Contribution Policy

The proposed measures for this category are:

- What is the Development Status of the software?
- What is the quality of the user documentation? Is a readme file included which demonstrates the basic usage of the software?
- Is test data made available?
- Is the software benchmarked against alternatives? Is it faster or slower?
- What is the quality of the developer documentation? Does the software provide API-level documentation?

There is an open question about whether a measure for ease of use should include something about whether cloud / one-click / containerised / bundled versions of a piece of software exist.

C. Maintainability

This category assesses the likelihood that the software can be maintained and developed over a period of time by measuring the effort associated of the project, the code quality.

The important metadata associated with this category include:

- Effort
  - Vitality: What is the average time between release cycles?
  - Activity: number of commits. discussions
  - Number of Contributors: including the trend (rising/stable/falling)
  - Number of Key Contributors: including bus factor[9] on each part of the software
  - Average time to fix bugs
- Code Quality
  - Code complexity
  - Unit test coverage
  - System tests

The proposed measures for this category are:

- What is the average time between major releases?
- Is the contributor community growing, staying stable or falling?
- What is the test coverage?

D. Portability

This category assesses the ability of the software to be used in a different area, particularly a different area of research, or to be commercialised.

Because this category is of higher perceived importance to research software than other types of software, there is less in the existing literature looking at potential measures and indicators of success.

[9] Bus Factor: https://en.wikipedia.org/wiki/Bus_factor

Intuitively, it would appear that measures for this category would be split between those giving an indication of whether the functionality of the software is useful to other areas / types of users, and those giving an indication of how easy it is for the software to be adapted and integrated. Therefore it is likely that this category is actually a superset of other indicators, and future work is necessary to identify if a combination of other measures can be used to predict the portability of a piece of software outside of the more common definitions of portability with respect to operating system or hardware architecture.

IV. Discussion

The previous section has set out a range of proposed measures which this author has identified as being of potential importance in enabling different stakeholders to assess research software. An emphasis has been placed on choosing measures where it can easily be identified if the software does not provide measurable information at all (i.e. no license, no documentation provided) as well as being relatively easy to measure where it does. As noted at the start, the aim is to make this something that developers of software will be happy to submit their software to for assessment, without fear of inappropriate comparison.

One way of addressing this is by designing the framework in such a way that it allows different communities to decide which indicators to use that best represent their requirements for comparing software. This also encourages more "bottom-up" adoption of the practice.

On the other hand, adoption could be driven by funders asking applicants to ensure that they have either chosen the software they propose to use based on some set of agreed measures (for instance license) or for software development projects asking them to reach a particular target measure before a second stage of funding will be released.

A challenge is that it is not obvious which of these measures can actually be used to provide an indication of software quality and longevity. This is because the "success" of a research software project is somewhat subjective: a piece of software which is no longer maintained may still be the main software used by a small niche of a research community to publish world-leading research.

Nevertheless, the only sensible approach is to attempt to categorise and assess current research software projects, in consultation with the community, to determine the correct categories that would help benchmark best practice in scientific software. This is being taken forward by a pilot study combined with development of tooling to help with the assessment by the Software Sustainability Institute in the second half of 2016.

A final question relates to the frequency of assessment. What are the right times to perform an assessment? In general, assessment would take place at least at the release of every major version of a piece of software. For some measures (for instance test coverage), which can be automatically generated, they can be incorporated into continuous integration systems. For research software, there are some points which may also have significance: the submission of a research paper; when applying for new funding; when reproducing an existing study or method.

Comments on and contributions towards the measures proposed in this paper are welcomed by the author.

Acknowledgment

The Software Sustainability Institute is supported by the EPSRC, ESRC and BBSRC through grant EP/N006410/1. This paper is based on input from many members of the Software Sustainability Institute team, Fellows, and collaborators including the participants of the CodeMeta workshop[10], Catherine Jones, and Patrick Aerts and Peter Doorn.

References

[1] EPSRC. 2012. *Software as an Infrastructure.* Accessed on 8th July 2016 from: http://www.epsrc.ac.uk/newsevents/pubs/software-as-an-infrastructure/

[2] Hettrick, S. 2014. *£840 million: the UK's investment in software-reliant research in 2013*. Accessed on 8th July f2016 from: http://www.software.ac.uk/blog/2014-10-22-840-million-uks-investment-software-reliant-research-2013

[3] UK Department of Health. 2013. Guide to creating a front of pack (FoP) nutrition label for pre-packed products sold through retail outlets.

[4] Holibaugh, R et al. 1989. *Reuse: where to begin and why.* Proceedings of the conference on Tri-Ada '89: Ada technology in context: application, development, and deployment. p266-277. DOI: 10.1145/74261.74280.

[5] Frazier, T.P., and Bailey, J.W. 1996. *The Costs and Benefits of Domain-Oriented Software Reuse: Evidence from the STARS Demonstration Projects.* Accessed on 21st July 2014 from: http://www.dtic.mil/dtic/tr/fulltext/u2/a312063.pdf

[6] CMMI Product Team, 2006. *CMMI for Development, Version 1.2.* SEI Identifier: CMU/SEI-2006-TR-008.

[7] Gardler, R. 2013. *Software Sustainability Maturity Model.* Accessed on 21st July 2014 from: http://oss-watch.ac.uk/resources/ssmm

[8] NASA Earth Science Data Systems Software Reuse Working Group (2010). *Reuse Readiness Levels (RRLs), Version 1.0.* April 30, 2010. Accessed from: http://www.esdswg.org/softwarereuse/Resources/rrls/

[9] Marshall, J.J., and Downs, R.R. 2008. *Reuse Readiness Levels as a Measure of Software Reusability.* In proceedings of Geoscience and Remote Sensing Symposium. Volume 3. P1414-1417. DOI: 10.1109/IGARSS.2008.4779626

[10] Chue Hong, N. 2013. *Five stars of research software*. Accessed on 8th July 2016 from: http://www.software.ac.uk/blog/2013-04-09-five-stars-research-software

[11] Bourque, P. and Fairley, R.E. eds., (2014) *Guide to the Software Engineering Body of Knowledge*, Version 3.0, IEEE Computer Society http://www.swebok.org

[12] ISO/IEC 25010:2011(en) (2011) Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models

[10] CodeMeta Participants: http://codemeta.github.io/

[13] Penzenstadler, B. and Femmer H. (2013) *A Generic Model for Sustainability with Process and Product-specific Instances*. In First Intl. Workshop on Green In Software Engineering and Green By Software Engineering.

[14] Microsoft Code Analysis Team Blog. 2007. *Maintainability Index Range and Meaning*. Accessed on 8th July 2016 from: https://blogs.msdn.microsoft.com/codeanalysis/2007/11/20/maintainability-index-range-and-meaning/

[15] Sjoberg, D.I.K et. al. 2012. Questioning Software Maintenance Metrics: A Comparative Case Study. In proceedings of ESEM'12. P107-110. DOI: 10.1145/2372251.2372269

[16] Ratnakar, V., and Gil, Y. 2015. Ontosoft. Accessed on 8th July 2016 from http://ontosoft.org/ontology/software/

[17] Crowston, K. et Al, 2006. Information systems success in free and open source software development: Theory and measures, Software Process Improvement and Practice, vol. 11, pp. 123-148.

[18] Subramaniam, C. et al. 2009. *Determinants of open source software project success: A longitudinal study*. Decision Support Systems, vol. 46, pp. 576-585.

[19] English, R., and Schweik, C. 2007. *Identifying success and abandonment of FLOSS commons: A classification of Sourceforge. net projects*, Upgrade: The European Journal for the Informatics Professional VIII, vol. 6.

[20] Wiggins, A. and Crowston, K. 2010. *Reclassifying success and tragedy in FLOSS projects*. Open Source Software: New Horizons, pp. 294-307.

[21] Piggott, J. and Amrit, C. 2013. How Healthy Is My Project? Open Source Project Attributes as Indicators of Success. In Proceedings of the 9th International Conference on Open Source Systems. DOI: 10.1007/978-3-642-38928-3_3.