J-Tracker - Sustainability Evaluation

This software evaluation report is for your software: J-Tracker. It is a list of recommendations that are based on the survey questions to which you answered "no".

If no text appears below this paragraph, it means you must already be following all of the recommendations made in our evaluation. That's fantastic! We'd love to hear from you, because your project would make a perfect case study. Please get in touch (info@software.ac.uk)!

Question 3.1: Is your software available as a package that can be deployed without building it?
Building software can be complicated and time-consuming. Providing your software as a package that can be deployed without building it can save users the time and effort of doing this themselves. This can be especially valuable if your users are not software developers.
You should test that your software builds and runs on all the platforms it is meant to support, which means you will already have created packages that can be distributed to your users!
See our guide on Ready for release? A checklist for developers (http://www.software.ac.uk/resources/guides/ready-release).
If you're interested in the consequences of ignoring your users' needs, see our guide on How to frustrate your users, annoy other developers and please lawyers (http://www.software.ac.uk/resources/guides/how-frustrate-your-users-annoy-other-developers-and-please-lawyers).

Question 4.5: Do you provide troubleshooting information that describes the symptoms and step-by-step solutions for problems and error messages?
Troubleshooting information helps users quickly solve problems. It can help to reduce the number of support queries that you have to deal with.

Question 4.8: Do you publish your release history e.g. release data, version numbers, key features of each release etc. on your web site or in your documentation?
A release history allows users to see how your software has evolved. It can provide them with a way to see how active you are in developing and maintaining your software, in terms of new features provided and bugs fixed. Software that is seen to be regularly fixed, updated and extended can be more appealing than software that seems to have stagnated.

Question 5.3: Does your project have an e-mail address or forum that is solely for supporting users?
E-mail is purpose-made for resolving users' problems. The user can provide a good description of their problem and can attach screenshots, log files or other supporting evidence, and it's easy for you to ask for follow-up information, if required. You should always try to have an e-mail address for support queries.

It’s best if the support e-mail address is clearly labelled as such e.g. myproject-support@myplace.ac.uk. This makes it easy for users to identify the e-mail address on your website or within your documentation, and it helps you to separate your support queries from all of your other e-mail. However, a personal e-mail address is better than nothing if you don’t have the means to provide a dedicated support address.

See our guide on Supporting open source software (http://software.ac.uk/resources/guides/supporting-open-source-software). Its advice applies to supporting closed source software too.

Question 8.1: Is your software cross-platform compatible? e.g. does it run under two or more of Windows, Unix/Linux and Mac OS X, or can be used from within two or more of Internet Explorer, Chrome, Firefox and Safari?

Choosing which platforms to support is a balancing act. The more platforms that you support, the wider your pool of potential users. But, the more platforms you support, the more you'll have to maintain, develop, test and support your software! Decide which platforms will deliver you the largest number of possible users, given the time and effort you have for development and maintenance. Always target your community’s popular platforms – don’t make them migrate to yours (as they won’t!)

Linux has a strong presence in the technical computing arena, e.g. on clusters, high-performance computing platforms and clouds. However, Linux systems typically require a greater set of skills, especially when it comes to building and deploying software. Microsoft Windows is the world's most widely used operating system, and it may be an important platform for your users. Even if your software has a Linux-specific server component, perhaps a Windows-based client would be a good idea. Developing using languages and products designed for, or runnable under, Windows may yield more efficient and easier to install software than insisting that users run your code under a Linux virtual machine.

Mac OS X might not be considered a core platform to support, given its relative low uptake compared to Windows and Linux based systems. But, ask yourself whether the Mac 'cool' factor has permeated your user community, and whether your software would only require small modifications to support it? Counting the number of Macs versus PCs at your community's favourite conferences might give you some pointers!

For browsers, no single browser has an overwhelming market share over the others, so if you want to maximise the uptake of you browser application, you should aim to support those with the largest market share: at least, Internet Explorer, Mozilla Firefox, Google Chrome and Safari.

Question 10.5: Do you back-up your repository?

While third-party repositories are very useful, it can be reassuring to know that you have a local back-up of your repository in case the third-party repository goes down for any length of time, or the host ceases to provide the repository hosting service. Distributed revision control repositories such as Git and Mercurial can be easy to back-up locally, just by cloning the repository. Centralised repositories such as Subversion or CVS rely upon the repository hosting service to provide you with the functionality to back-up your repository (see, for example SourceForge Subversion back-ups (http://sourceforge.net/p/forge/documentation/svn/#backups).

Question 11.8: Do you have tests that can be run after your software has been built or deployed to show whether the build or deployment has been successful?

Tests give a user confidence that your software has built and installed correctly on their platform. If a test fails, the nature of the failure can help the identify why e.g. maybe the user forgot a

configuration step, or provided an incorrect configuration option.
For developers, tests contribute to a fail-fast environment, which allows the rapid identification of failures introduced by changes to the code such as optimisations or bug fixes.
Tests are an important aspect of maintainable software. There are many frameworks available for writing tests in a range of languages, including JUnit (http://junit.org/) for Java, CUnit (http://cunit.sourceforge.net/) for C, CPPUnit (http://www.freedesktop.org/wiki/Software/cppunit/) and googletest (https://code.google.com/p/googletest/) for C++, FRUIT (http://sourceforge.net/projects/fortranxunit/) for Fortran, py.test (http://pytest.org/) and nosetests (http://nose.readthedocs.org/) for Python, testthat (https://cran.r-project.org/web/packages/testthat/index.html) for R and PHPUnit (https://phpunit.de) for PHP.
Alternatively, you can provide a list of steps that a user can take to check the deployment of the software e.g. for a web-based application, this might just be checking that it the application is accessible via a browser.

Question 12.1: Do you have an automated test suite for your software?

After changing your code and rebuilding it, a developer will want to check that their changes or fixes have not broken anything. Tests contribute to a fail-fast environment, which allows the rapid identification of failures introduced by changes to the code such as optimisations or bug fixes. The lack of tests can dissuade developers from fixing, extending or improving your software, as developers will be less sure of whether they are inadvertently introducing bugs as they do so.
Each test might verify an individual function or method, a class or module, related modules or components or the software as a whole. Tests can ensure that the correct results are returned from a function, that an operation changes the state of a system as expected, or that the code behaves as expected when things go wrong.
There are many frameworks available for writing tests in a range of languages, including JUnit (http://junit.org/) for Java, CUnit (http://cunit.sourceforge.net/) for C, CPPUnit (http://www.freedesktop.org/wiki/Software/cppunit/) and googletest (https://code.google.com/p/googletest/) for C++, FRUIT (http://sourceforge.net/projects/fortranxunit/) for Fortran, py.test (http://pytest.org/) and nosetests (http://nose.readthedocs.org/) for Python, testthat (https://cran.r-project.org/web/packages/testthat/index.html) for R and PHPUnit (https://phpunit.de) for PHP.
Automating the run of your test suite means the entire set of tests can be run in one go, making life easier for your developers. Having an automated build system is a very valuable precursor to providing a test suite, and having an automated build and test system is a valuable resource in any software project.
See our guides on Testing your software (http://software.ac.uk/resources/guides/testing-your-software) and Adopting automated testing (http://github.com/softwaresaved/automated_testing/blob/master/README.md).

Question 12.3: Do you use continuous integration, automatically running tests whenever changes are made to your source code?

Having an automated build and test system is a solid foundation for automatically running tests on the most recent version of your source code whenever changes are made to the code in the source code repository. This means your team (and others if you publish the test results more widely) obtain very rapid feedback on the impact of changes. Continuous integration servers can automatically run jobs to build software and run tests whenever changes are committed to a source code repository. For example, Jenkins (http://jenkins-ci.org) is a continuous integration server that

can trigger jobs in response to changes in Git, Mercurial, Subversion and CVS. Travis CI (http://travis-ci.org) is a hosted continuous integration server that can trigger jobs in response to changes in Git repositories hosted on GitHub (https://github.com).
See our guides on How continuous integration can help you regularly test and release your software (http://software.ac.uk/how-continuous-integration-can-help-you-regularly-test-and-release-your-software), Build and test examples (https://github.com/softwaresaved/build_and_test_examples/blob/master/README.md) (which includes walkthroughs on Getting started with Jenkins and Getting started with Travis CI), and Hosted continuous integration (http://www.software.ac.uk/resources/guides/hosted-continuous-integration).
Going further, this can also be done automatically whenever the source code repository changes. See our guides on Testing your software (http://software.ac.uk/resources/guides/testing-your-software), Adopting automated testing (http://github.com/softwaresaved/automated_testing/blob/master/README.md)

Question 13.1: Does your project have resources (e.g. blog, Twitter, RSS feed, Facebook page, wiki, mailing list) that are regularly updated with information about your software? (e.g. release announcements, publications, workshops, conference presentations)
These resources are all good ways of showing that your project is active. If potential users see frequent posts, especially if they talk about new features and resolved problems, they know that your project is thriving, your software is useful and is under active development. This may encourage them to use your software, knowing that if they run into problems, there may be people who can help, and who they can share experiences with.
These resources also give you a way of getting in touch with your current users, keeping them engaged, and asking for their input or help with problems. Don’t know what new feature to implement next? Ask your users whether they think it would be useful.

Question 13.2: Does your website state how many projects and users are associated with your project?
Where you have an active set of users and developers, advertising their existence is not just good for promoting the success and life of your project. If potential users see that there are a large number of users, they know that your project is thriving, your software is useful and is under active development. This may encourage them to use your software, knowing that if they run into problems, there may be people who can help, and who they can share experiences with.

Question 13.3: Do you provide success stories on your website?
A great way of showing off your software is to write case studies about the people who've used it and how they've used it. This helps potential users learn about the software but, more to the point, is a great advert for your software. If you can show happy users benefiting from your software, you are likely to gain more users.

Question 13.4: Do you list your important partners and collaborators on your website?
Providing a list of important partners and collaborators gives potential users valuable assurance that your software has a future. Also, the higher the scientific, academic or industrial reputation of those partners, the higher the perceived reputation of your software, and project, will be.
Publicly recognising partners' efforts in improving or working with your software also increases the likelihood they will continue to use, or develop, your software in the future. Credit where credit is

due!

Question 13.5: Do you list your project's publications on your website or link to a resource where these are available?
Listing your software publications provides an academic perspective on the value of your software. It can also help users, and other stakeholders (e.g. current and potential funders) to understand, in detail, how your software contributes to research, what scientific problems it has helped to solve. In addition, these can help to show where your software sits in relation to other software that fulfils a similar need, and what makes yours different, or better.
These publications also gives researchers something to cite when they write their own papers where your software has been used, which is of value to them and also increases your citation count for your papers, which helps you demonstrate your impact!

Question 13.6: Do you list third-party publications that refer to your software on your website or link to a resource where these are available?
Providing a list of third-party publications can show, academically, how the software is used by others, as well as promoting their efforts and successes.
It also gives potential users ideas for how they may choose to use the software, as well as providing assurance that the software can be used by people other than its original developers to achieve something. Having such a list also means you can cite these publications in your own papers, funding proposals and reports to show or justify its value and the impact you have made!
As a matter of routine, you should always ask people to cite your software if they've used it in their research for these reasons, and to inform you if they have included such a reference in one of their papers.

Question 13.7: Can users subscribe to notifications to changes to your source code repository?
Keeping information on these notifications as open as possible helps you present the impression that your project is open and inclusive. If users are actively developing using your software, keeping them up to date with on-going development on your source code (e.g. implementation of enhancements, extensions or bug fixes) enables them to factor these into their own development plans. For example, users might want to use an up-to-date copy from the repository to benefit from bug fixes made since your last release. If the notifications include information on the changes made (e.g. excerpts of the source code showing additions, removals and alterations) then this may allow for rapid identification of bug as any subscriber may notice an issue in the changes made.
A lightweight, automated subscription process can reduce, even remove, from you the overhead of managing notifications. It also means that users aren't subject to long delays waiting for their membership to be approved.

Question 15.1: Does your website and documentation clearly state the copyright owners of your software and documentation?
A clear statement of copyright for your software and documentation provides a clear message to anyone using your software, who created, and owns, the software and documentation. As such, anyone visiting your website or reading your documentation will know exactly who to contact if they have a question about using, modifying or redistributing your software and documentation.
It is also essential for users and developers to know the copyright of any third-party software bundled in a release for the same reasons. This can include: third-party source code, copied in and used as-is; modified, extended, or bug-fixed, third-party source code; third-party binaries (e.g.

DLLs, JAR files etc) shipped by you; and, third-party software that is downloaded and installed by users.

Question 15.2: Does each of your source code files include a copyright statement?
It’s easy to distribute source code files, and this separates the code from any copyright statement that might be on your web site or in your documentation. To cover this eventuality, and remove any ambiguity about ownership, it's good practice to include a copyright statement with each of your source code files, as a comment, or, if the language permits it, as a string constant.

Question 15.6: Does each of your source code files include a licence header?
It’s easy to distribute source code files, and this separates the code from any licence statement that might be on your web site or in your documentation. To cover this eventuality, and remove any ambiguity about what a developer can do with the source code, it's good practice to include a licence statement within each of your source code files, as a comment. This can also help to avoid confusion between source files that may have different licences, particularly if there are a number of third-party dependencies used within your software.

Question 15.7: Do you have a recommended citation for your software?
Asking that users cite your software, either directly or via its associated publications, provides you with credit for develop your software. It also provides a means, via harvesting of citations, of gathering evidence of the uptake and exploitation of your software. See, for example, Citing R (https://cran.r-project.org/doc/FAQ/R-FAQ.html#Citing-R) and Citing Taverna (http://www.taverna.org.uk/cite/).
See our guide on How to cite and describe software (http://software.ac.uk/so-exactly-what-software-did-you-use) and examples of the citations recommended by various software packages (http://www.software.ac.uk/blog/2014-07-30-oh-research-software-how-shalt-i-cite-thee).

Question 16.2: Does your website or documentation describe how your project is funded, and the period over which funding is guaranteed?
Especially on academic projects, users will view the active lifetime of software to be the duration of the software’s project funding. If you want to persuade users that your software will be around in the future, it is a good idea to describe your funding model and the duration over which funding is assured.