Harmony - Sustainability Evaluation

This software evaluation report is for your software: Harmony. 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 1.2: Does your website and documentation clearly describe the type of user who should use your software?

Very little software is designed for users from all backgrounds. If your software is only designed for users with specific types of knowledge or expertise (both research domain-related and software-related), you must be upfront about this in your documentation. If you are not, you risk attracting users who will find your software confusing or unsuitable for their needs, and who may complain about this fact or request more support than you want to provide.

Question 1.3: Do you publish case studies to show how your software has been used by yourself and others?

A great way of showing off your software is to write case studies about how yourself, and others, have used it. Case studies can help potential users learn about your software. They also act as a great advert for your software. If you can show happy users benefiting from your software, you are likely to gain more users.

Question 2.1: Is the name of your project/software unique?

You shouldn't choose a project or software name that is shared by another group – especially if they are a competitor – but, sadly, few people spend enough time researching the uniqueness of their project or software names. It is, now, less important to have a domain name that mimics the name of your project or software, because most people will use a web search to find a website rather than manually entering a URL. However, it is important to check, using a web search, that there are no existing domains that include the name of your project or software that are owned by another group, because this can confuse your users.
See our guide on Choosing project and product names (http://software.ac.uk/resources/guides/choosing-project-and-product-names).

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 5.4: Are e-mails to your support e-mail address received by more than one person?

It's easy to forget about an e-mail, especially one that's asking difficult questions, so your e-mails to your support e-mails address should always be received by more than one person. One person should still have the primary responsibility of handling users' e-mails, but others can step up to handle e-mails if necessary, so that a user's query will be acknowledged even if one of you is on holiday, ill or otherwise indisposed.

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 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 12.2: Do you have a framework to periodically (e.g. nightly) run your tests on the latest version of the 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 at regular intervals e.g. nightly. At its simplest, this can be done by scheduling a cron job on Unix/Linux or Mac OSX, or using Windows Task Scheduler. A more advanced solution is to use a framework like Inca (http://inca.sdsc.edu/) which harnesses a number of machines through a central server to distribute a wide variety of tests in a parallel and scalable way.

See our guide on Testing your software (http://software.ac.uk/resources/guides/testing-your-software).

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 13.8: If your software is developed as an open source project (and, not just a project developing open source software), do you have a governance model?
A governance model sets out how a, open source project is run. It describes the roles within the project and its community and the responsibilities associated with each role; how the project supports its community; what contributions can be made to the project, how they are made, any conditions the contributions must conform to, who retains copyright of the contributions and the process followed by the project in accepting the contribution; and, the decision-making process in within the project.
Though they are designed for open source projects, many of their concerns are relevant to any software project.
OSS Watch (http://oss-watch.ac.uk) provide an introduction to governance models (http://oss-watch.ac.uk/resources/governancemodels).

Question 14.2: Do you have a contributions policy?
A contributions policy provides information to users on what they can contribute (e.g. bug fixes, enhancements, documentation updates, tutorials), how they can contribute it (e.g. via e-mail, patch files, GitHub pull requests), any requirements they must satisfy (e.g. compliance to coding or style conventions, passing required tests, software licensing), and what happens to their contributions once they have submitted it (e.g. how it is reviewed and then integrated into your code, documentation or web site). It also tells users who owns the copyright on their contribution.
For information on how to manage contributions, see OSS Watch's Contributor Licence Agreements (http://oss-watch.ac.uk/resources/cla).

Question 14.3: Is your contributions' policy publicly available?
Users may not contribute if they do not know that they can contribute. Publishing your contributions policy provides information to users on what they can contribute, how they can contribute it, any requirements they must satisfy, and what happens to their contributions once they have submitted it. It also tells users who owns the copyright on their contribution.
For information on how to manage contributions, see OSS Watch's Contributor Licence Agreements (http://oss-watch.ac.uk/resources/cla).

Question 14.4: Do contributors keep the copyright/IP of their contributions?
Asking contributors to sign over their copyright and intellectual property to your project or organisation can put off users from contributing. It, in effect, asks them to give away ownership of something that may be novel and which may represent a key aspect of their research. Allowing contributors to keep their own copyright and intellectual property removes this barrier, thereby making contribution a more attractive option. It also contributes to promoting a community round your software – everyone sharing their outputs rather than handing them over to a select group.
For information on how to manage contributions, see OSS Watch's Contributor Licence

Agreements (http://oss-watch.ac.uk/resources/cla).

Question 16.1: Does your website or documentation include a project roadmap (a list of project and development milestones for the next 3, 6 and 12 months)?

A roadmap allows users to see when new features will be added and plan their project accordingly. It also has an important secondary benefit: one of the most important factors that will influence a user's choice of software is the likelihood of that software still being around – and supported – in the future. There are many ways in which a project can persuade a user of its longevity: regular announcements, regular releases, prompt replies to queries, information about funding and its plans for the future – a roadmap.

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.

Question 16.3: Do you make timely announcements of the deprecation of components, APIs, etc.?

It's never a good idea to remove components or features without giving your users an advance warning first. It could be there are users who are dependent on the feature(s) you plan to change or remove. Announcing such planned deprecations well in advance means users and developers can respond if a given feature is important to them.

If a feature is due to be superseded by a newer, better feature or component, including both for a suitable period within the software can allow your users to transition comfortably from the older version to the new version.

You could also consider developing and publicising a deprecation policy, stating how and when features or components in general are deprecated. This gives your users assurance that features will not be removed without warning. see, for example the Eclipse API deprecation policy. (https://wiki.eclipse.org/Eclipse/API_Central/Deprecation_Policy).