Adopting Open Source Software for Longer-term Reuse

Steve Crouch, SSI
s.crouch@software.ac.uk

Advanced School on Scientific Software Development: Concepts and Tools

24/02/2011

Background

- Looked at sustainability in your own software – let's look at it from the other side…
- Assume you have software that needs a new feature
- You could implement the feature yourself
  - Control over the feature's implementation
  - No unneeded code or features
- Or, could make use of other software that has the feature, e.g. a code library
  - It exists already, no re-inventing the wheel
  - Make use of other features (if helpful)

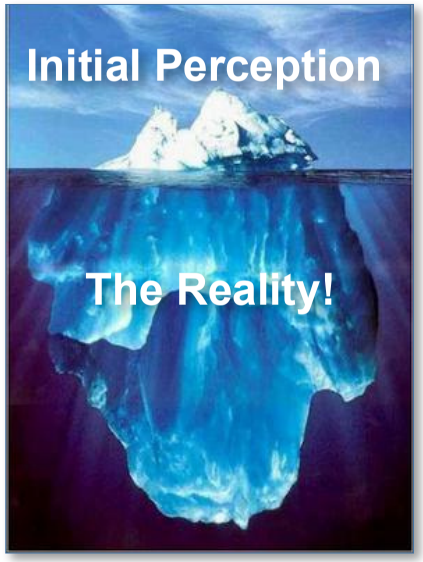
- *Informed reuse* should be encouraged
  - Can save you time and effort now and later

Picking the Right Software

www.software.ac.uk

- Often many open source choices out there
  - e.g. on SourceForge
- But, should consider time required to
  - Learn new software
  - Test it!
  - Integrate new software into your own
  - Manage updates to adopted software e.g. security, fixes
- Choosing the wrong software can be expensive!

- Taking time early to make a good choice can save you a lot of time later
  - Assume you've found a likely candidate piece of software
  - Let's start by answering a few questions about it…



Effort required to adopt third-party software

Picking the Right Software

- Easy to use the ‘nearest’ software library
  - Installed on machine already
  - It’s the first candidate you’ve come across
  - You’ve heard about it
  - Unfortunately, it’s common practice…
  - …although it could be the right choice
- Taking time early to make a good choice could save you ***a lot*** of time later
  - Assume you’ve found a likely candidate piece of software
  - Let’s start by answering a few questions about it…
- Goal: try to make an *informed decision for reuse*

Does it Do What You Want?

- Yes, it's an obvious question!
- What are your requirements
  - Now, but also in the future… project goals?
  - Does it go far enough?
  - If it doesn't quite fit your needs, can it be (easily) extended?
- What does your community use/need?
  - Do they use an established piece of software for this?
- Are any needed features planned for deprecation?

Is the Software Right for its Role?

- How will the software help to fulfil a goal?
- Investigate prior uses of the software
  - Look for use cases or case studies e.g. on website
  - Have other projects used it as you need to use it?
- Technically…
  - Evidence the software will integrate with your own
  - Track record for reliability
  - Test the software in its operating context

Is it Actively Used, Developed and Supported?

- Importance of *active* community (and its size) shouldn't be underestimated
  - User community
  - Developer community
- Look for evidence of development
  - Code repository (SVN/CVS check-ins)
  - Releases, feature enhancements
- Look for evidence of support
  - Forums, issue tracker - actively used, good level of response?
  - Direct email
- Structured development
  - Process for development?
  - Code contribution policy?
  - Deprecation policy?

Does the Software Have a Future?

- If development/support for software ends…
  - e.g. lack of funding, or software has been superseded
  - Could find yourself with unsupported software
  - Leads to software decay
- Will software exist within project's lifecycle (and beyond)? Often difficult to estimate…
  - Ask them!
  - Development, support and community
  - Roadmap? Previous history of frequent releases? Planned releases?
  - Actively updated project website?
- Open standards
  - Has your community converged on appropriate standards?
  - Does software support them where you'd expect?
  - Evidence of established interoperability with other implementations?
  - e.g. Keyhole Markup Language (KML) for geographical annotation and visualisation, grid open standards from Open Grid Forum (OGF)

How is the Software Provided?

- Good documentation!!!
  - User, administration, development (e.g. API)
- Take into account your own skills
- Pre-requisites…
  - Easy to obtain and deploy?
  - Fit your own requirements e.g. operating system support
- If developing with the software…
  - Access to source code repository e.g. SVN, CVS?
  - Understandable, modular, easy to build?
  - Unit tests or test suite?
- Licensing
  - Do you have right to use/distribute it for its intended purpose?
  - Does software respect third-party copyright and licensing for its own dependencies?

Ask the Developers!

- Why not ask the developer community any important questions about the software?
- A sign of good support if the developers are responsive and friendly
- Asking a developer can give you
  - Indication of the type of responses you can expect from them in the future
  - As well as filling in any knowledge gaps to make your decision

Example: Taverna

- Workflow management system, for designing and executing workflows
- Started as a system for bioinformaticians, its use has expanded to include many other disciplines
- Comprehensive user, developer and API documentation, roadmap
- Source code under revision control in Subversion with a contribution licensing policy (Contributor License Agreement)
- Extensibility framework
- Active and responsive bug tracking system, regularly updated website
- Supports a host of domain-specific and web service standards,, development funding until 2014
- Organised training and workshops
- Long list of example uses of Taverna

- http://www.taverna.org.uk/

Example: JGraphT

- Java library for producing graphs
- User support and announce mailing lists, source code under revision control (SVN)
- Feature and bug trackers with high activity and prompt responses
- User Wiki for all things JGraphT
- Code contribution policy and good API documentation

- http://www.jgrapht.org/

- *Example shows that you probably won't get **all** the right answers*
- *Need to decide what **is** important for your project*

Further Resources

- SSI guides:
  - http://www.software.ac.uk/choosing-right-open-source-software-your-project
- Other resources:
  - OSS Watch: http://www.oss-watch.ac.uk/resources/tips.xml