Report on current software and practices (task 1)

NMI-3 Workpackage 6 FP7/NMI3-II project number 283883
November 2012 - R. Leal and E. Farhi

Table of Contents

Abstract

In this report, we have evaluated a selection of data analysis software for neutron scattering experiments. The practices used to develop and maintain the software are also analysed in order to define a set of recommendations to be used in further projects. This reports fulfils the Task 1 of the work-package.

The criteria used for the software evaluation are Deployment / installation, Usability, Functionality, Maintenance and Expandability. The criteria used for the software practices are related to version control, points of failure, testing, documentation, and code duplication.

A Software for neutron data analysis

The neutron scattering software selected for this analysis was mainly those distributed in the ready to run LiveDVD [http://nmi3.eu/about-nmi3/other-collaborations/data-analysis-standards.html]. See Table 1 for the list of software evaluated.

We chose to group the software in the following fields of research (ordered from the highest number of users to the lowest):

1. Diffraction
    1. Powder
    2. SANS
    3. Single-crystal
2. Spectroscopy
    1. Time-of-flight
    2. Triple-axis spectrometer
3. Reflectometry
4. Backscattering
5. Spin-echo
6. General use, that is software that cover more than one field, or can be used independently of the type of data set.

Table 2 classifies the software according to the field of research.

A.1 Software development status

Table 3 illustrates the overall status of the tested software. A fair part of the projects are not active any longer, others are merely active for bug-fixes and just a few appear to be actively developed (e.g. Mantid, Sasfit/sansview, McStas, iFit, LAMP, FullProf suite, Vitess). Mantid is to our knowledge the only one that involves a serious community of developers of about twenty full time active developers.

The majority of the software is hosted in code repositories websites. **TODO ?**

A.2 Software OS and Installation

All the software evaluated in this report were tested under Linux Ubuntu 12.04 operating system – the same operating system present on the live CD available on the NMI3 website. According to the web sites where those applications can be downloadable, all the software can run in the three principal operating systems (windows, Mac OS X and Linux). However, for Linux and according to our tests, not all binary files can be executable out of the box. Normally due to back compatibility issues, the already compiled software written in C++ or Fortran may have issues with binary shared libraries (i.e. libstdc++ and libgfortran).

This problem is actually present in any modern Linux version. It arises when users try to run a legacy program that was compiled against and old shared library (in an older Linux version), usually, libstdc++ or libgfortran. This can usually be fixed by recompiling again the software from the source (when it is available). In our opinion, however, this may put off prospective future inexperienced users.

Another issue, that the inexperienced users may face under Linux and Mac OS X, is the amount of extra libraries required from some programs. Windows packages also suffer from this dependency issue, in a lower extend, by relying on DLL files which may be system dependent. The only practical solution for developers to overcome this issue is either to reduce the number of dependencies or bundle the software with all the necessary dependencies. The latter may be impracticable due to the size of some external libraries.

A few software distribute the Linux version as RPM or/and DEB packages. Those packages are usually capable of calculating dependencies and fetch transparently the necessary libraries from the internet before installation. However, these packages are often Linux version dependent, and not all versions are supported by the developers.

Bundled software, such as LAMP or iFit, are distributed in a single package including all external dependencies and, regarding its size, may be beneficial for users with limited computer skills. Lamp for example, has a live update feature, which fetches the last version from the internet and updates the program transparently to the users.

A.3 Software programming features

Several programming languages and libraries are present in this study. As expected the majority of the legacy programs are written in procedural languages such as Fortran. Not only in the context of this study, but in general, software that started to be developed in the 70-80s, are mostly Fortran based. Despite the widespread use of modern technologies to wrap Fortran code and create bindings in scripting (interpreted) languages with little programming effort, some active software packages are still developed in Fortran (e.g. CrysFML library and FullProf Suite).

Proprietary frameworks are also present in this report. IDL was a platform of choice in the 90s for scientific development. LAMP and DAVE are two programs that use that language. The Matlab language is also used, as two recent projects are based in this platform (iFit and Grasp). These software can be run without purchasing the Matlab/IDL platform, but advanced development may require a purchased licence. Rather high inherent costs can thus prevent a certain number of possible software developers to contribute to the code. Yet, LAMP and Grasp, for example, still feature a large community of users. The simplicity and power of iFit starts to attract a significant number of more experienced users used to Matlab platform.

Java code is quite unusual in scientific environment. In this study, only ISAW platform and the triple-axis instrument simulator vTAS were implemented in Java. ISAW developers added Jython scripting language support to facilitate the customisation of ISAW at other laboratories. Jython has the same syntax as python, however it does not provide support for other python packages as Numpy or Scypy.

Python tends to be indeed the favourite programming language among scientists for recent software. (e.g. GSAS-II and GenX). The simplicity of python programming allied to scientific packages (e.g. scipy, matplotlib, which mimic many features of proprietary packages such as Matlab) has made python scripting very popular.

However, python as interpreted language performs usually slower than compiled languages (e.g. Fortran, C or C++). It is generally accepted than python performs between 4 and 100 times slower

than C++. Some packages compiled in, usually, C++ and C (e.g. numpy) and integrated in python, can help improve performance when used to store and manipulate large datasets. Yet, recent software, such as Sansview and Mantid, have taken the decision to develop the core infrastructure in C++ and build python bindings to allow users/scientists to contribute and write their own scripts in Python. In Mantid, some of the components such as GUIs and algorithms are indeed written in Python.

The Frida software has been developed in C++ and has recently migrated to the most recent version of the standard of the C++ programming language (C++11). Although this might create some issues with old version of GCC, C++11 introduces new features to facilitate the software development. We believe that the C++/Python combination might be wide spread in the next coming years.

Some analysed software is still coded in procedural programming languages (non object oriented, i.e., clear separation between data and functions) such as C and Fortran. Despite the simplicity of the development and the easiness to keep track of program flow, procedural languages lack clear modular structure and the abstract data types where implementation details are hidden, which in turn reduces extensibility.

A great portion of the analysed code started to be developed a few decades ago, to facilitate and automate certain tasks. Due to the continuous requirements for new features, these programs have grown on the same basis (i.e., unstructured, design-less). Although the presence of this legacy code (robust code proven by decades of usage and debugging) does not represent a problem, still developing new functionalities upon this paradigm at present can be seen as unnecessary effort and an increased additional complexity. The software developed following this approach use lexically and syntactically complex languages (e.g. Fortran). It is usually agreed that the non-use of object oriented programming (OOP) make the code unstructured, difficult to read and extend, and very risky to modify.

Despite these drawbacks, there are still ongoing development in procedural languages (e.g. Sasfit, McStas, FullProf Suite). It is clear that the lack of object oriented design increases the difficulty to understand what are the main functions of the code and exposes unnecessary features, which otherwise would be abstracted. Notwithstanding, some of these software developers tend to organise the software in folders to keep the functionalities organised.

Mantid appears to be the unique project that follows a strong object-oriented design. Although the Mantid design was inspired by the GAUDI (http://proj-gaudi.web.cern.ch/proj-gaudi/) platform at the LHC- Geneva, nowadays both architectures are quite different from each other. Mantid recoded some of the concepts from Loki library (http://loki-lib.sourceforge.net/) and POCO library (http://pocoproject.org/) and takes advantage of boost smart pointers (http://www.boost.org/). Several “Design Patterns” (Abstract Factory, Proxy, Command) from the book “Design Patterns: Elements of Reusable Object-Oriented Software” are implemented. The Template definition and specialisation (Abstract Factories and Singletons) is also observable within the main components. In Mantid, however, the object-oriented design appears to be pushed to the extreme. In some peculiar situations, Mantid overuse class heritage where class composition would perhaps be a better solution. Very often one can see several levels of heritage making the code quite complex to understand. This is a known problem of OOP: over-use (or indeed abuse) of inheritance when composition is clearly superior for object designs.

A.4 Usability and GUIs

It is a fact that the neutron sources are being visited by a growing number of non expert users. In our opinion, very few software seem adequate to this type of users. Non expert users often concentrate on the scientific problem in question and not on the technical details of data processing and evaluation. None of the analysed software, except LAMP, presented more than one possible user interface (e.g. normal and expert mode).

This fact led us to think that the majority of the software described here is designed by an isolated scientist (or a few...) who tend to work in small and focused groups. The collaboration or interaction with others (special possible future software users) tend to be very limited. The majority of the oldest software was built and maintained by a highly skilled single person and there is limited knowledge sharing. This can be seen as a single point of failure in the software life-cycle.

The frequent result of this methodology is a very complicated software to use, developed without thinking on the inexperienced users. The developer appears to assume that most users share the same degree of knowledge.

Just a limited number of software, if any, appears to include non-expert users in the requirements and developing process. Mantid, for example, has been including the feedback of instrument scientists in the developing process and just now (5 years after the start of the project) is thinking of integrating the feedback of some users in the development tasks. Often, this feed-back happens in the find-error/fix bug and new feature requirement by mean of email exchange.

A great part of the tested software provide a GUI interface. Exception occurs for Frida, PDFfit, iFit and GSAS. The latter has no GUI, but a graphical user interface (EXPGUI) is available as a separate program.

Java programs, as expected, use the Java native Swing library for interface development. The programs built under proprietary software use the native GUI system (e.g. Dave, LAMP). Some “old style” primitive interfaces are coded in TK (either through TCL or Perl). The FullProf suite uses a limited commercial platform called Winteractor. New GUIs are rather implemented in the python library wxPython (SANSView, PDFGui, GenX, GSAS-II).

Despite the popularity of Qt in the IT community, only Mantid features a GUI based on this library. The Qt toolkit is a cross-platform application framework mainly for graphical user interface. It is natively built in C++ but provides bindings for other languages, including python. To the authors knowledge, this library is very powerful but has a steep learning curve, making wxPython a very attractive alternative for scientific software developers.

Almost all of the tested software possesses plotting facilities. Those based in Python often use matplotlib (GenX, SansView, GSAS-II). Frida for example uses Gnuplot. Sasfit uses TCL/TK blt tool kit. Java software uses java native plotting libraries. Mantid was built as part of QtiPlot and uses its integrated library for plotting. It also links to Vates (a customised version of Paraview) for 3D visualisation. Commercial platforms use native plotting facilities.

It is worth noting that LAMP has a server side application running at the ill.fr website. It exposes remotely the main functionalities of the software through an HTML interface. To our knowledge, in addition to LAMP, only McStas and vTAS provide a web interface. The Mantid team also starts to

consider a rich internet application for a close future. This interface will be more limited than the current one (for security issues, no python scripting interface should be available).

B Practices of the software developers

As opposed to the software built by software engineers, scientific software is simply a means to an end rather than the ultimate aim. Such software is used as a tool to progress in research. As a consequence, scientific software thus lacks of requirements, architecture design and documentation, which is mandatory in any commercial IT product. The scientific software is usually very specialised to a particular topic and is rarely extensible or interoperable.

The scientific software based on software architectures using a set of standards common to enterprise architectures usually fail (e.g. DANSE). These projects typically take too long to develop and suffer from poor adoption. For scientists, requirements are emerging and constantly evolve during the whole project. It is thus difficult for software engineers to capture requirements and design a software solution to be used by scientists.

Mantid appears to be the closest to an enterprise software solution. It is managed by a private company and counts roughly 20 active developers. The coding teams are based in UK and US are releasing new features on a short basis, as suggested by agile development principles. For developers, the internal complexity of Mantid, added to the large number of contributors, brings maintenance issues. Such concerns did not arise in any past software, which were reduced in size, yet very effective with the same scientific knowledge. This may indicate that the overall size of Mantid is artificial and brings little gains compared to other previous simpler solutions.

B.1 Coding and Hosting

Good practices start to arise. The great majority of the software analysed is hosted by code repositories (SVN, Mercury or GIT protocols) with commit tracking features (see table 3) .

Few exceptions arise for code that is not freely available: GSAS, vTAS, and part of the Fullprof Suite are not available for download. Some source code, although available for download through the source repository, are not freely available, such as in GSAS-II: “GSAS-II routines are copyright protected, but are available for reuse in other non-commercial codes with appropriate scholarly acknowledgement”. Most projects adopt a GPL-like licensing scheme.

Despite the development of some software on proprietary development frameworks (IDL, MatLab, IGOR and PV-wave), the code is available for download. Although the development on such commercial platforms typically implies the payment of licence fees, the learning curve is usually not very steep and the scientific tools provided are often seen as a great advantage.
However, we do not think these tools are very adapted for large scale projects. For instance, Matlab object-oriented design performance has a very bad reputation. It appears that there may be, depending on the programming methodology, a substantial method call overhead (higher than mainstream OO languages) making Matlab not an optimum solution for OOP. Yet, we do think that the code produced on these platforms is of great value if properly incorporated as components in other applications. Both Matlab and IDL provide run time shared libraries that can be accessed through other languages. Given the amount of efficient and well tested legacy code available, we

think that this option should be considered for future developments. Also, it is granted that the learning curved required to code with a high-level language such as Matlab and IDL is assumed to be definitively shorter than with a more performance-effective lower-level language such as C++. In total, there is not 'best guess' coding solution, and any software is a complex equilibrium between coding, maintenance and performance costs.

B.2 Testing

The process of testing and refining software appears to have been forgotten. Almost none of the software reviewed possesses a UnitTest platform. Exception arises for Mantid which, according to the best practices, a test must be written for every new functionality. Mantid uses the google testing platform, including Mock tests.
iFit, although not implementing any specialised UnitTests platform, has a set of test routines. The same happens with GSAS II and McStas. Other software provide example script files that can be seen as tests.

Mantid possesses two Jenkins (http://jenkins-ci.org/) Continuous Integration Servers (performing builds on Windows 32 & 64 bit, Linux & Mac) that perform an automatic build of the Mantid Framework, MantidPlot and the install packages following each check-in to the Mantid Github repository. The developer is notified if he or she breaks the build or if the test fails. McStas uses a simpler similar package for developers notification.

B.3 Documentation

User manuals are often available. Some of them are rather occasional guides than exhaustive step by step guides though. iFit, for example, provides exceptional documentation for both beginners and advanced users. The source code also appears to be well documented. GSAS-II, despite the not-intuitive interface provide good tutorials for less experienced users.

Some of the code is not intuitive and lacks documentation both in the code and technical documentation that describes the source code. Comments in the code generally sparse when existent. Some of the commits text to source repository are not very informative either. It is clear that much has to be done in this area.

Mantid is a typical case of large multi-year, -site, and -million dollar infrastructure project. It started with a wiki and good documentation, but the will to keep the documentation growing and up to date appears to be lost. It is still visible on the wiki deprecated features and functionalities that are not available any more and can mislead inexperienced developers or contributors.

Some software (including Mantid, Sasfit... ???) use auxiliary software, such as Doxygen (http://doxygen.mantidproject.org/), to generate browsable code documentation. Although useful to navigate through the code and the class dependencies figures (when existent), if the code is not properly commented, the value of this solutions is very limited.

Mantid appears to be the only software presented here that had a software architecture planned. However the situation to date is rather different from the initial plan and documentation about the current architecture is missing. The last documentation available about design dates from 2009.

McStas also started with a careful architecture design, which has been kept robust since then.

B.4 Code reuse and duplication

Re-factoring and reusing existing code is a quite general concept nowadays. On the recently developed software present in this study, two techniques were widely used: 1. the complete recode of old applications in a new programming language and 2. a “facelift” to the user interface and introduction of new features keeping the main core of the application (legacy code) unchanged.

The most “shiny” software available to date are either based on the legacy source code with new interfaces (e.g. EXPGUI interface for GSAS) or full recode of the old application. In GSAS II, for example, only 5% of the legacy Fortran code was kept. For PDFfit2 the decision was to completely rewrite the old Fortran-77 PDFfit engine in C++, and create python bindings to facilitate the production of specific routines and bindings. Mantid was fully built from scratch using the QtiPlot interface. All the scientific algorithms have been recoded in C++ using the Gnu Scientific Library (GSL).

Our experience with recent software supports the opinion that a fresh new software will never perform as better as an old software with 20 or 30 years of testing and fixing. Attempts to “re-invent the wheel”, as Gumtree and DANSE have failed in past and recoding a new solution is still considered very risky.

Code duplication and replication is evident throughout this analysis. Duplication of features appears to increase with the developers number and not only with the project size. Mantid for example uses a tool called CPD in its integration server to probe for code duplication. The result of running this tool shows a non negligible amount of duplicated code. Probably due to more than twenty active developers coding (somehow independently) in different places of the world.

It is no surprise to find common, and thus overlapped, functionalities in different software. These functionalities are often rewritten in different styles, and thus not imported (even if they are freely available).

Not only from this analysis, but it is common place to see different scientific groups developing concurrently the same solution. Sometimes, forking and customising existing software (e.g. Sassena at the SNS, an nMoldyn fork). Often, the new solution is worst than that the existent. One may thus argue that collaboration and contribution to a solid software package could be a better solution.

This led us to conclude that collaboration among groups must be strengthen. In the style of the CERN ROOT package, one could envisage to list all current software exportable functionalities so that new software could directly choose these as libraries. Such a catalogue could list models, algorithms, I/O routines, interface design templates, ...