NMI-3 Workpackage 6 FP7/NMI3-II

project number 283883

Task 1: Review existing data analysis software and practices of software developers

A number of data-analysis software were tested and rated according to the following functionalities:
Appreciation:

- Deployment / installation
- Usability
- Functionality
- Maintenance
- Expandability
-

Points to discuss:

  - code repository
  - Single man work
  - documentation (science and code)

This report aims to analyse the most common software of neutron data analysis as well as the practices of the scientific software developers. A software list is presented on the first part of this report whilst on the second more technical aspects are reviewed.

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 ~~opted~~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.

Software development status

Table 3 illustrates the overall status of the software tested. 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, McStas, iFit, LAMP, FullProf, Vitess). Mantid is to our knowledge the only 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

Software OS and Installation

All the software evaluated on this report were~~as~~ 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, M~~AC O~~s~~x~~ac OSX 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 usually present in any modern Linux version. It arises when users try to run a legacy program that was compiled (in an older Linux version) against and old shared library, 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 inexperienced users may face in Linux and Mac OS X is the amount of extra libraries required from some programs. Windows packages also suffer from the dependency issue, in a lower extend, by reliying on DLL files which may be system dependent. The only practical solution for developers to overcome this issue is to reduce the number of dependencies.

A few software ~~are~~ distribute the Linux version as RPM or/and DEB packages. Those packages are usually capable of calculating~~ed~~ 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 usually supported by the developers.

Bundled software, as LAMP or iFit, are distributed in single package including all external dependencies and, regarding its size, may be beneficial when 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.

Software programming features

Several programming languages and libraries are present in this study. As expected the majority of the ancient 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 ~~the~~a platform of choice in the 90s for scientific development. LAMP and DAVE are two programs that use that language. Despite the costs of Matlab licences, two recent projects are based in this platform (iFit and Grasp). Both software can be run without purchasing the M~~m~~atlab platform, but ~~any~~ serious development may require~~s~~ a purchased licence. Rather high inherent costs can thus prevent~~ing~~ 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 experience users used to Matlab platform.

Java code is quite unusual in scientific environment. In this study, ~~just~~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~~s~~ 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.

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 interchange might be wide spread in the ~~recent~~next coming years.

Some software analysed are still coded in procedural programming languages 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 code analysed 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 ~~several years~~decades of usage and debugging), does not represent a problem, still developing new functionalities upon this paradigm at present ~~it is~~may appear as a great mistake. This software is usually built using lexically and syntactically complex languages, such as Fortran (e.g. crysFML), which make the code unstructured, difficult to read and extend, and very risky to modify. Ongoing development in procedural languages is also observable (e.g. sasfit, mcstas), but then makes use of modern infrastructure. 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. The code duplication is also apparent.

Mantid appears to be the unique project that follows a strong object-oriented design. Although Mantid design was inspired by the GAUDI (REF) platform at the LHC- Geneva, nowadays both architectures are quite different from each other. Mantid recoded some of the concepts from Loki library (REF) and POCO library (REF) and takes advantage of boost smart pointers (REF). Design patterns from GoF book (REF) are implemented as templates and specialised for the main components.
However, object-oriented design appears to be taken to the extreme. In some ~~particular~~peculiar situations, Mantid overuse class heritage where~~n perhaps~~ compositions 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.

Usability and GUIs

It is a fact that neutron sources are being visited by a growing number of non expert users. In our opinion, very few software seem~~s~~ adequate to this type of users. Non expert users concentrate usually on the scientific problem in question and not on the technical details of data processing and evaluation. None of software analysed, 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 ~~are~~is built 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 high skilled single person and there is no knowledge sharing. This can be seen as a single point of failure in the software lifecycle.

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, 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 ~~after~~years after the start of the project) is thinking of integrating the feedback of some users in the development tasks.

A great part of the software tested provide a GUI interface. Exception occurs for Frida, PDFfit, ~~Ifit~~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. 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).

Full prof suite uses a very limited commercial platform called winteractor.
New GUIs are rather implemented in the python library wxPython (Sansview, PDFGui, GenX, GSAS2).
Despite the popularity of Qt, only Mantid features a GUI based on that 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.
It worth noting that LAMP has a server running at the ill.fr website and has an HTML interface for running as a remote application. Some other packages also provide a web interface (McStas).~~The same happens for GRASP (??? any others?)~~

Almost all software tested possesses plot facilities. Those based in Python usually 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 uses Vates (a customised version of Paraview) for 3D visualisation. Commercial platforms use native plotting facilities.

Practices of the software developers

Contrary to software built by software engineers, scientific software is simply a means to an end rather than the ultimate aim. Software is used as a tool to progress in research. Scientific software thus lacks of requirements or architecture design or documentation. The software is usually very specialised to a particular research and is rarely extensible or interoperable.

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 t~~d~~o capture requirements and design a software solution to be used for 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.

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 XXX) .

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”.

Despite the development of some software on proprietary development frameworks (IDL, MatLab, IGOR and PV-wave), the code is available for download. As said before, costs incurred for the development licence fees may not stimulate collaborative and pro-active development. *[je ne suis pas d'accord, car le cout d'apprentissage afin de contribuer a Mantid est bien supérieur a celui d'une license Matlab, IDL ou Igor]*

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.

Mantid uses the Jenkins platform (REF) which runs all tests every time a user pushes its code to the remote repository. The user is notified if he or she breaks the build or if the tests fail. McStas also has such a feature.

Documentation

User manuals are usually available. Some of them are rather occasional guides than exhaustive step by step guides though. iF~~If~~it, 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.

Mantid was 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.

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. Mantid was the 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 replication is evident throughout this analysis. 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). This led us to conclude that collaboration among groups must be strengthen.

Code duplication in the same project is also observable. Mantid for example uses a tool called CPD for probing 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 in different places of the world....

When collaboration and contribution to a solid software package could be conceivable, fork of the project is perceptible (e.g. Sassena @SNS (nMoldyn fork).

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, ...