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~~reviewed 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 and ~~partially covers~~aspects of ~~the~~ Task 2.

The criteria used for the software review~~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]. ~~from the software projects which are easily available on the internetThis selection is obtained~~ . See Table 1 for the list of~~reviewed~~ software ~~reviewed~~~~evaluated~~.

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

Modily the table towards either instr or science. If ranking, cite source for criteria.

1. ~~Diffraction~~Structure
   1. Powder
   2. ~~SANS~~Large scale structures (SANS)
   3. Single-crystal
2. Spectroscopy
   1. Time-of-flight
   2. Triple-axis spectrometer
   3. Muon
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 categories above~~the field of research~~.

Define simulation, reduction and analysis categories. Detail what is a 'general use' soft.

Below is an attempt at definition for simultion, reduction and analysis....:

In addition to group the reviewed software according to the type of beamlines/science it has been applied to, the software may also be categorized according to whether has been applied to simulation, reduction or analysis. However, scientists in different area of science and even within the same area of science associate different meanings to such categories. Here is what we mean by these, in relation to software for neutron scattering and muon spectroscopy:

- Simulation: Software which has been used to simulate data from a virtual instrument, where this virtual instrument is either a software representation of an existing neutron or muon instrument or such an instrument in the process of being designed.
- Reduction: Software that takes raw data collected on a beamline and remove instrument/detector specific artifacts from the data, including detector efficiencies variations over a detector bank, sample can scattering etc..
- Analysis: Software which takes data collected at a beamline, raw or processed, and infer some physical quantity from the data which is dependent on sample used for the experiment.

More detailed list for SANS focusing on redundancy.

Xmas present report.
Send Table 2 to authors for comments.
Appendix with package list which gives a list of instruments where it is used. Table 4.

A.1 Software development status

Table 3A and table 3B 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 fitthteen twenty full time active developers.
Split T3 to be able to see all items

A.2 Software OS and Installation

All the software evaluatedreviewed 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 downloaded, 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 an old shared library expected to be part of the operating system (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 usersHaving to compile from source makes the deployment of a a software harder.
Another issue, that the inexperienced users may face under Linux and Mac OS X, is the amount of extra libraries required for some programs. Windows packages also suffer from this dependency issue, in a lesser extent, 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 for 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, s~~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 on 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. Also, ~~The simplicity and power of~~ iFit has started~~s~~ to attract a significant number of more experienced users accustomed to the Matlab platform.

Java code is less commonly used in the neutron scattering and muon spectroscopy ~~in thequite unusual~~ scientific environment. In this study, only the ISAW platform and the triple-axis instrument simulator vTAS were implemented in Java. ISAW developers added the 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, such as the popular Numpy or Scypy.

Python tends to be indeed the favourite programming language among scientists for recent software (e.g. GSAS-II, GenX and many others not covered in this report). 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. A great effort is being devoted to port scientific packages to Python (RPy for the R Project for Statistical Computing, SymPy for symbolic mathematics, Biopython for biological computation, etc.).

However, Python as an interpreted language performs usually slower than compiled languages (e.g. Fortran, C or C++). It is generally accepted that~~n~~ Python performs ~~between 4 and 100 times~~ slower than C++ but is faster for prototyping. 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. Python and C++ integrates is supported, for example, by the Boost Python binding library, and advantages of large scale scientific software based on a combination of C++ and Python has been advocated for a while, see e.g. *Journal of Applied Crystallography* 2002, **35**, 126-136, *Acta Cryst.* 2002, **D58**, 1948-1954, http://phenix-online.org/cctbx/. In the neutron scattering and muon spectroscopy community ~~Yet, recent software, such as~~ Sansview and Mantid~~,~~ have follow this approach also. D~~taken the decision to d~~evelop 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 coming years.

Some analysed software is still coded in procedural programming (PP) 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.~~
Object Oriented Programming (OOP) can be advantageous over PP as both data and functions are wrapped into clear modular entities (objects). Moreover, OOP code is easy to maintain and modify as new functionalities can be be created (extended) with small differences to existing ones. All in one, the benefits of OOP can be summarised as: abstraction, encapsulation, modularity, polymorphism, and inheritance.

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~~)~~ makes the code ~~unstructured, difficult to read and~~ more difficult to extend, and thus very risky to modify.

T~~Despite these drawbacks, t~~here 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.~~ S~~Notwithstanding, s~~ome 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. T~~Although t~~he 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 ~~recode~~uses the~~some of the concepts from Loki library (http://loki-lib.sourceforge.net/) and~~ POCO library (http://pocoproject.org/) and many parts of the ~~takes advantage of~~ boost ~~smart pointers~~library (http://www.boost.org/).
Several “Design Patterns” (Abstract Factory, Proxy, Command), as for example described in ~~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~~The Mantid code design is not static and is either as a consequence of a review or new use cases it is continuously modifying. ~~.~~ Currently, it is the authours opinion that i~~I~~n some parts of the code~~peculiar situations~~, Mantid overuse class heritage where class composition would ~~perhaps~~ be a better solution. At such places~~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, as in the GOF book: "Favor object composition over class inheritance."~~.~~

Highly related to computer technology history.

A.4 Usability and Graphical User Interfaces

~~Table 3: define GUI columns symbols '++'.~~ ~~N~~It ~~is a fact that the n~~eutron source facilitie~~s~~ are being visited by a growing number of non expert users. ~~In our opinion, very few software seem adequate to this type of users.~~ Any~~Non expert~~ user ~~should~~s ~~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~~,~~ present~~sed~~ more than one possible graphical user interface that is~~(e.g.~~ normal and expert mode~~)~~.

T~~his fact led us to think that t~~he majority of the software described here is designed by an ~~isolated~~ one or a few scientist~~s~~ ~~(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.

It is the authours opinion that t~~T~~he 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 development process. For Mantid ~~, for example,~~ a mechanism for getting requirements is through scientific steering commetees consisting mainly of instrument scientists. Instrument scientists are expert user, although they represent the user community of their instruments, which includes non-expert users. Hence, for example for the ISIS Muon instruments, a GUI interface was designed for non-expert Muon users. The Mantid project has more recently been working on coming up with an alternative mechanism for directly interact with non-expert users. It is not as easy as to get information from non-expert users because they are not so readily available and not so easy to engage. An alternative email exchange and conferences/workshop where users may report bug fixes request and new feature. The experience for Mantid is request are either from expert users and non-expert users who are developers wanting to evaluate, create a new loader, integrate other software etc with Mantid. ~~requirement by mean of email exchange. bug and new feature in the find-error/fix feed-back happens has been including the feedback of instrument scientists in the development 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~~

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.

Mantid features ~~two separate and independent components: the Mantid~~ one f~~F~~ramework, which is exposed through a Python interface~~a standalone library with python bindings~~, which allow users to load and process data through a Python console, useful for expert-users; and ~~2. the~~ MantidPlot, a GUI to the Mantid framework which is used by both by expert and non-expert users. ~~data analysis and scientific visualisation solution to interact with the Mantid Framework~~.

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 older~~"old style" primitive~~ interfaces are coded in TK (either through TCL or Perl). The FullProf suite uses a limited commercial platform called Winteractor. Newer GUIs have been~~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 P~~p~~ython. 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. MantidPlot 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 the near 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 for a particular topic and is rarely extensible or interoperable.

Building~~The~~ scientific software based on ~~software architectures using a set of~~ standards and common ~~to~~ enterprise architectures does not guarantee alone a successful outcome, ~~usually~~ **~~fail~~** ~~(~~e.g. DANSE~~).~~ d~~D~~id not complete initial design, although parts of the project~~but~~ produced valuable software. Large scientific software~~These~~ projects therefore need to typically take too long to develop and suffer from poor adoption. For scientists, requirements are emerging and constantly evolving~~e~~. Therefore regardless of the size of a project producing useful software at an early stage of the project is important ~~during the whole project~~. Also, it may be difficult ~~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. The project has gained from the expertise of a specialised scientific software consultancy company both at a project management level and coding level. ~~It is managed by a private company and counts roughly 20 active developers.~~ The coding team~~s~~ is~~are~~ based in UK and US and ~~are releasing~~ new features are released on a regular basis, as suggested by agile development principles.

Scientific software of any size can have maintainess issues. In the case of Mantid this risk is reduced by a suit of unit tests and system tests, at the half yearly developing meetings discussing

parts of the code developers have found particular painful and way to improve on this, having various diagnosis reported as part of each incremental build and days where all developers gets involved in removing compiler warnings or improving code documentation.

~~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 gain 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 curve required to code with a high-level language such as Matlab~~, and~~ IDL or Python is ~~assumed to be definitively~~ shorter than with a more performance-effective lower-level language such as C++ or Java. In total, there is no obvious coding solution, and any software is a complex equilibrium between coding, maintenance and performance costs. It all depends on the initial design, and the programmers ability to keep it simple yet efficient.

B.2 Testing

~~The process of testing and refining software appears to have been forgotten.~~ Almost none of the software reviewed possesses a u~~U~~nit t~~T~~est or system test platform, where unit tests refers to for example testing of methods in classes, and system tests test to see if a sequence of steps done by the software produces an expected result. ~~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 unialising~~implementing~~ any specialised u~~U~~nit t~~T~~ests platform, has a set of test

routines. The same is true forhappens with GSAS II and McStas. Other software provide example script files that can be viewedseen 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 automatedic builds of the Mantid Framework, MantidPlot and the install packages following each check-in to the Mantid Github repository. AThe developer is notified if he or she breaks the build or if the tests 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 very good 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 are generally sparse when they exist. 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. For Mantid there is roughly two sets of documentations. One is for the benefit of the developers, which include Doxygen documentation and wiki pages. The other, and is documentation aimed at users, which is partially written by developers and partly by users of the software. For example, introductory documentation for using the non-expert Muon interface within MantidPlot is entirely maintained by the Muon scientist who for example use this for running Muon workshops. The quality of the user documentation created by the developers, is largely determined by the stakeholders and users of the project, who determine the overall task priory list of the project. Hence the user documentation of the Mantid project has seen improvements based on the demand of it users. The current user documentation has been accepted for users at Mantid and SNS and ISIS. However, recent interest in Mantid outside SNS/ISIS has highligthed the lack of documentation in a number of areas. It started with a wiki and good documentation, but the will to keep the documentation growing and up to date appears to be lost. Still visible on the wiki are deprecated features and functionalities that are not available any more and can mislead inexperienced developers or contributors.

Some software (including Mantid and Sasfit, Frida) 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 existing), if the code is not properly commented, the value of this solution is very limited. For Mantid this kind of documentation has been for useful for developers and a few expert users.

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. For the case of~~On~~ the recently developed software, reviewed~~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. The MantidPlot interface of Mantid is ~~was fully~~ built on top of ~~from scratch using the~~the QtiPlot application~~ interface~~, and the fit engine in Mantid uses the ~~. All the scientific algorithms have been recoded in C++ using the~~ Gnu Scientific Library (GSL) fitting tools.

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 appear~~s~~ 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).

Mantid chose to use algorithms as mean to extend the Framework. This plugin architecture through the Factory pattern is very convenient for the extensibility of the platform. However, the form that the algorithms have been implemented encourage code duplication. Common code to several algorithms is usually reused through hierarchy, and when algorithms need to implement functionalities that are in other algorithms either they call the concerned algorithm or the code is duplicated. A great part of algorithms are enormous structure of code with a great part, if not all, of the functionality incorporated in the main class. A better option would be to keep the code in the algorithms mininal and implement its functionality (or perhaps that that could be common to other algorithms) in separate packages.

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 worse than that the existing software. One may thus argue that collaboration and contribution to a solid software package could be a better solution.

Our recommendation is~~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 exporting?? ~~able~~ functionalities so that new software could directly choose these as libraries. Such a catalogue could list models, algorithms, I/O routines, interface design templates.~~,~~ ~~.....~~

Rework this.

TO ADD (or not):

##As a user:
###Positive:
- All the plotting facilities
- Curve fitting
- Vates
- GUIs to deal with the certain instruments

###Negative:
- Doesn't carry the info from one WS to the other (e.g. energy, wavelength)
- However it keeps in the forms the last text typed.
- For IN5 for example, it creates cash files of 1.2Gb in `.mantid/`, making the PC very slow:
```-rw-r--r-- 1 leal lss 1.2G Mar 15 14:47 WS2D2.tmp0```
- The workflow of algorithms is tedious and confusing without notes.
```