Report on the evaluation of the prototype software (Task 4 D6.4)

NMI-3 Work-package 6 FP7/NMI3-II project number 283883
Feb 4th, 2015 - E. Farhi (with input from members of the work-package)

Version 0.22

Abstract

This report documents the Task 4 of the Data Analysis Standards work-package (NMI3-II/WP6). It deals with the evaluation of the prototype software that was produced during this project.

Table of Contents

1. Introduction

During the Data Analysis Standards work package (NMI3-II WP6), most of the production phase was focused at demonstrating that the Mantid project could be used for continuous neutron source instruments. As a result, 7 Mantid instrument data loaders, 6 new treatment algorithms (correction/reduction), and 6 minor corrections to existing algorithms were produced. In addition, 9 instrument detector geometries were described. The produced code is available in the Task 3 report D6.3, as well as on the work-package web page <http://nmi3.eu/about-nmi3/networking/data-analysis-standards.html>, and is now mostly included in the Mantid release. In this report, we focus on the code contributed to the Mantid project <www.mantidproject.org>. Two other minor projects developed during the WP6, the *reductionServer* and the *AllToMantid* algorithm, will not be considered here. Also, it is out of the scope of this document to benchmark the accuracy/efficiency of the Mantid algorithms compared to other software. We assume that, in case of any difference, once the necessary algorithms have been produced, the maintenance work would allow to identify and possibly correct/explain these.

2. Mantid code metrics

We perform a code analysis of the Mantid project <http://mantidproject.org> using standard metrics tools, mostly relying on the code size, number of lines of code, number of contributors, etc. We assume that most of the scientific, instrument focused, contribution resides in the Framework (Algorithms and underlying objects), as well as Scripts, instrument definitions and possibly Vates. This is indeed where the 'scientific' part resides, and where the loading, and reduction steps are stored. In the following we shall refer to the 'Algorithms' as the Framework, scripts, Vates and instrument definitions. For the analysis presented below, we used the Mantid git repository as of Oct 31st 2014.

The Mantid project was started in 2008. We can thus estimate the daily production for the duration of the 7 years (with 250 working days/year) up to now as 840 LOC/day, 48.5 kb/day and 2.7 files/day for the eligible 'Algorithms' contribution part listed in the Table 1, and all contributors. The corresponding whole Mantid production is 1307 LOC/day. The total number of LOC increases rather linearly with time since 2008, as 341 kLOC/year. Also, we can compute the mean LOC length as 59 characters and 64 characters for the 'Algorithms' and the whole Mantid project respectively.

Mantid part	*LOC*	*Size (kb)*	*files*
Framework	995455	48136	3928
Scripts	75561	10752	272
Vates	32711	2352	297
Instruments	366314	23624	276
Total 'Algorithms'	**1470041**	**84864**	**4773**
Total Mantid Code	**2287130**	**143964**	**6556**

Table 1: Selected Mantid parts analysis (Oct 31st 2014 git) using <cloc.sourceforge.org>. The LOC is the number of lines of code (including comments and blanks). A kb is 1024 bytes.

According to the OpenHUB Mantid page <https://www.openhub.net/p/Mantid>, the total number of contributors in Mantid is given as 77 contributors (38229 commits) since 2008, 47 contributors in the last 12 months (9108 commits), and 26 contributors (697 commits) in the last month. The development rate has clearly increased since the origin of the project in 2008. Since 2011, the mean number of contributors in Mantid is about [20-25] continuous coders. The analysis of the production of each contributor shows that 64 % of the total code has been produced by the 10 most prolific coders. The Mantid code base grows regularly of about 341 kLOC/year.

There is no easy way to find the exact number of contributors for the 'Algorithms' part corresponding with the Table 1. As an approximation, we shall use the ratio of the listed Mantid parts LOC to the total Mantid LOC, that is 1470041/2287130=0.64. Similarly, when using the size ratio and the file number ratio, we get estimates of 0.59 and 0.72 respectively. Using the intermediate LOC ratio 64% estimate, we can derive an effective number of about [20-25] *64%=[13-16] contributors for the Algorithms, scripts, instruments and Vates. Each contributor cost can be estimated using a 60 keuros/year/contributor cost (rather on the low side), that is 242 euros/working day/contributor, including taxes.

Then, we draw the number of lines of 'Algorithms' code produced per contributor per day as 840/

[13-16] = 52-64 LOC/day/contributor, 3-3.7 kb/day/contributor, and 0.16-0.21 file/day/contributor. Merging these estimates, we derive a Mantid 'Algorithm' cost per line of code of 3.7-4.6 euros/LOC.

The current cost on the whole 'Algorithms' part can be inferred as [13-16]*242 = 3.1-3.9 keuros/day, and the whole Mantid cost is 4.8-6.0 keuros/day, or 1.2-1.5 Meuros/year (using the 64% ratio). Integrated on the whole Mantid life-time and using the total number of LOC for the Mantid project, we derive an estimate of about 8.4-10.5 Meuros, which is to be compared with the CoCoMo cost estimate of 29 M$ computed with the OpenHUB service.

Let's now focus on the functionalities provided by Mantid, and more specifically on the mean effort required per Algorithm available for the end-user. The number of 'public' algorithms listed on the Mantid documentation <http://docs.mantidproject.org/nightly/algorithms/index.html> is currently 626. These functionalities include for instance loaders, data correction/reduction, general operators, input/output filters, instrument specific methods, … In addition, the import routines are designed in conjunction with an instrument definition file, which specifies the geometry of the detector pixels. Summing up these parts, we divide the results in Table 1 by 626 and draw that every 'public' Algorithm functionality requires about 2348 LOC, 135.56 kb and 7.6 files, which corresponds to 36 to 45 coding work days and 8.8-10.8 keuros.

Last, we may estimate the maintenance effort per file by analysing their history. We point out that maintenance is a major task in the life of a software. Bugs have to be solved during the whole software life-time. We look at how many commits are needed per Algorithm for its 'support'. Often, the creation phase is visible as regular commits, then the commits become sparser. We make an analysis of the Framework/Algorithms commit dates and number (based on 20 different Algorithms code history, chosen randomly), and find that in Mantid the mean number of commits to create an algorithm is 9 spanning on about a year, and its maintenance phase, up to now, corresponds with 6 commits (in the following 2 years after the initial development year). As the maintenance is never finished, we have to be cautious with these estimates. However, once created, the maintenance can only increase, reducing as a side effect the creation capability of the work force, as long as the new code production goes on. Some bugs are easy to fix, some others may require a long search and even a full rewrite of lower level parts.

From these considerations, if we assume that an initial coding year resulted in 9 commits, then the next years will require about 3 commits every year. The maintenance level per year is then around 30% of the initial development effort. This estimate may be over-estimated, especially as one may consider that 'maintenance commits' involve less work than 'development commits', but it seems realistic to assume that it is above 20% for the Mantid project. In practice, maintenance is never ensure to be 'easy'. Some bugs may be easily fixable, but other require in-depth search and large re-coding. This 20% number corresponds with a software life-span of 5 years, which means that after that time, the existing developers would mostly maintain the existing software. Any new functionality would require new man-power.

Mantid code metric	*Value*
Daily production, lines of code	840 in 'Algorithms', 1307 total in LOC/day
Active contributors (2012-2014)	13-16 in 'Algorithms', 20-25 total
Daily production, per contributor	52-64 LOC/day/contributor
Cost per day, current (60 keuros/man-year)	3.1-3.9 keuros/day in 'Algorithms', 4.8-6.0 total
Cost per LOC	3.7-4.6 euros/LOC
Effort per 'public' algorithm	2348 LOC, i.e. 8.8-10.8 keuros, i.e. 36-45 days
Maintenance work ratio	About 20-30% of initial effort, per year

Table 2: Metrics for the Mantid 'Algorithms', which include the Framework, scripts, instrument definitions and Vates.

3. Work package WP6 Data Analysis code metrics

The Data Analysis Standards work-package (NMI3-II WP6 <http://nmi3.eu/about-nmi3/networking/data-analysis-standards.html>) focused on an evaluation of current software used in the neutron scattering community. It mostly focused on the Mantid project. R. Leal was hired to code Mantid algorithms needed to support continuous neutron source instruments. The project spanned over 27 months, from which 20 months were devoted to Mantid to produce 7 loaders, 6 new treatment algorithms (correction/reduction), and 6 minor corrections to existing algorithms. In addition, 9 instrument geometries where described. The supported instruments are IN16b, D2b, D33, IN4, IN5, IN6 at the ILL, MiBemol@LLB, Focus@SINQ/PSI. In addition, the IN10, IN13, and IN16 loaders at the ILL were contributed out of the NMI3-II project. Last, 3 lower level data object modifications were proposed to support 'scanning' instruments (such as diffractometers and TAS machines), but were not retained as they resulted in large instabilities of the whole existing Mantid infrastructure. These contributions are detailed in the NMI3-II WP6 Task 3 report (D6.3).

In total, 77437 lines of code have been pushed (166 commits) into Mantid, that is about 186 LOC/day. The code production has thus been higher than the mean Mantid corresponding metric.

We point out that only importers and basic data reduction were produced for the above instruments. It is expected that some of the other existing Mantid algorithms can be used further to process the created work-spaces after importing data files. However, this does currently not hold in the case of the 'scanning' diffraction instruments that were considered during this work package, in which case specific algorithms have to fully coded for most of the data treatment work-flow (see below).

We have collected data files from the ILL cycle 2014/3 produced by the instruments IN10, IN13, IN16 (cycle 2013/1), IN16b, D2b, D17, D33, IN4, IN5, IN6. We have installed the 'nightly' Mantid package dated Dec 2nd, 2014, and attempted to import these data files in MantidPlot. However, files could not be imported for 5 of these 10 instruments (IN10, IN13, IN16, D17, IN16b) from which 2 (D17 and IN16b) are handled by loaders from this WP6 project. The D2b data file could be imported as an 'MD' group, but can not be plotted nor used by any other existing Mantid algorithm. This test result is that 2 data file formats (D17,IN16b) loaders out of 7 (IN16b, D2b, D17, D33, IN4, IN5, IN6) are not functional (i.e. 28%). For D2b, nothing could ever be done with data loaded into a Mantid MD workspace – it was not functional before and it is not now (brings failure ratio from 28 to 42%).

We conclude 28-42% of the Mantid ILL loader algorithms are not functional after the funding of the WP6 project stopped in September 2014, that is 3-4 months. Assuming an exponential decay law,

we conclude that the mean algorithm half-time is 3-7 months.

In order to estimate the work needed per instrument, we analyse the Mantid python scripts produced during the NMI3-II WP6 for a few instruments. We find that these scripts call from 6 up to 14 different algorithms in order to produce reduced data as currently obtained with LAMP <http://www.ill.eu/fr/instruments-support/computing-for-science/cs-software/all-software/lamp/>, not including the geometry description. Looking at the LAMP 'SIM cards', we find that each instrument requires about 4-7 calls to specific functionalities. We then draw that about 6 methods per instrument is a conservative estimate of the effort per instrument.

With the current Mantid software hierarchy, it was found that it is difficult to import a series of non overlapping data acquisitions using the same detector in different locations, such as when using e.g. some scanning powder diffractometers and triple-axis spectrometers on continuous neutron sources. This type of data sets can be loaded into Mantid as an MD workspace, with one of the dimensions being the scan step, but the existing Algorithms can not be used to treat that data, not event to visualise it in a sensible way. Consecutively, the reduction and analysis algorithms would need to be written specifically for each of these data storage representations. Alternatively, the importation as a group of acquisitions did not succeed as the instrument description must then, in the current Mantid framework, be common to all data sets, which is not the case when the instrument moves. In the case where the scan configuration used during the acquisition is always the same (e.g. 25 scan steps on D2b at the ILL), it would be possible to describe a large 'virtual' detector, and fill it iteratively with each separate scan step to reconstruct the final full acquisition.

NMI3-II WP6/Mantid code metric	*Value*
Mantid Algorithm half-life time without maintenance	3-7 months (rough estimate)
Mantid Algorithm effort per supported instrument	6 'algorithms', 14 kLOC, 51-64 keuros, 0.86-1.08 man-year

Table 3: Refined metrics for the Mantid 'Algorithms' obtained from the WP6 produced code.

4. Mantid evaluation: Developer point of view

A modern, large software project

The Mantid project satisfies most of the recommendations listed in our Task 2 report D6.2. It provides a source version control system (Git), stored on the free (yet commercial) GitHub servers. It makes use of a collaborative platform (currently Trac, emails and Skype/Slack), has a very large set of unit/integration tests and automatically builds installation packages (using a Jenkins integration server). Each new code commit has a review process (using Git pull requests) which is evaluated within the developer team. The documentation is automatically generated and pushed on the project wiki pages.

We point out that these infrastructure recommendations are not specific to Mantid, as most modern projects present similar features (nMoldyn, McStas/McXtrace, LAMP, iFit, PyGSAS, BornAgain, SasView...).

In addition, Mantid provides developer specific tools. An extensive documentation on how to contribute, to set-up a developer system, to use coding standards, is available from the project web page. One significant feature is that end users can submit python scripts and algorithms to a dedicated, easily accessible upload area, which highly facilitates contributions without needing a

full authentication process as a Mantid developer.

On the developer side, python binding have been implemented to most C++ user level classes, which greatly help in writing python algorithms to contribute e.g. on the exchange upload area. These bindings come with examples, and there are scripts to generate the skeleton for new contributed Algorithms.

Code size and complexity

The current Mantid code base is larger than 2 MLOC, as indicated in Table 1. As a consequence, the compilation time is significant, e.g. 5 to 10 minutes as reported by the Mantid Jenkins server at <http://builds.mantidproject.org/>. This also means that the development procedure can become tedious with iterative changes and compilations, until new code is functional. This time may be reduced by splitting the project into independent sub-projects, requiring targeted and faster compilations upon code change. The test procedure is split into system tests, which last for about 3 hours, and performance/local tests which typically last about 10-20 minutes. As testing is part of the contribution procedure, it may in some cases represent a limiting factor in the code production.

Every change, including small bug fixes, must follow the same procedure: create a ticket, build locally, test locally, build and test in the build servers, pass tests by other developers, and eventually merge in the master branch. This procedure is necessary for such large projects to ensure at all time a stable repository, but can span on very long periods of time. In practice, we have in some cases experienced waiting of the order of a month between an initial change request and its final appearance in the Mantid master repository.

Due to the large code base, replicated code is common along the project. The Algorithms code could be made smaller by generalising the use of shared auxiliary functionalities, sorted by e.g. technique so that they are easy to find when starting a new Algorithm. With time, many technique and even instrument specific Algorithms have been implemented, whereas often some of these Algorithms could be made more general so that they can be re-used widely by other Algorithms. In the same spirit, Standard Operating Procedures (SOPs) should be advertised to avoid for instance redefining π (and other constants) in many places. The code architecture has reached a high level of complexity, with extensive hierarchical class levels, and many similar yet different classes that could in some cases be merged.

Last, the configuration of a Mantid development system requires the installation of about 30 third-party libraries, some of them being difficult to find for some systems. For instance, installing the Vates/Paraview bindings on a Debain/Ubuntu 14.04 system (current long term support) was found impossible due to unsolved library conflicts. MantidPlot currently requires a specific outdated ParaView version (version 3.98 customized by the Mantid team), incompatible with any other ParaView installation (currently version 4.3).

A way to solve most of these issues is to strengthen communication between developers, as well as enriching the coding standards and documentation for developers. General chat rooms, as used today, may be complemented with more dedicated chat rooms focusing on specific issues. The status of the infrastructure organisation and code hierarchy should be discussed to trigger code re-factoring on a regular basis (e.g. developer meetings), with assigned coders.

5. Mantid evaluation: User point of view

The Mantid project uses the MantidPlot user interface, which derives from QtiPlot <http://www.qtiplot.com/>, an alternative project to e.g. Excel, Origin, SigmaPlot, and Igor Pro. QtiPlot is a commercial product. This interface was initially developed as a spreadsheet-type

application, with plotting capabilities for 2D/3D graphs, and native python scripting.

The Mantid infrastructure provides a mechanism to automatically generate dialogue boxes from an Algorithm class. This way, most dialogue boxes benefit from a unified rendering which may be appealing to non-expert users. A large effort has been achieved end of 2014 to modernise the documentation, which provides help for every public Algorithm, including example use from the script level Python console.

As stated above, users can also contribute any file (e.g. data file, python scripts, …) into a shared upload area, specifying the author, and the description of the contribution. This feature is an excellent way to foster collaboration and exchange, but still requires a significant level of knowledge regarding both python and the Mantid infrastructure.

The installation package is large, and it requires a large number of 3rd party libraries to be installed. As a consequence, the installation takes a significant amount of time to complete, and may seem complex on some systems.

The user interface includes by default an extensive list of functionalities which may not all be needed by the end user. A solution to this problem may reside in the existing capability to configure the appearance of the MantidPlot interface (choose the default facility, the algorithm categories to show, as well as the visible toolbars). This feature should be advertised, and the authors may even distribute specific facility-oriented packages with pre-defined simple configurations.

The rendering of 2D/3D plots inherited from QtiPlot are somewhat limited compared to other commercial products as IDL, Origin, Igor, Matlab. This is particularly visible in the 3D plot rendering. The external plotter through ParaView, when properly installed, shows a simplified version of the ParaView engine which is more adapted to end-users, but remains much more complex than for instance a pure VTK surface/volume renderer as implemented in nMoldyn 4.