Deliverable Number:	D10.1
Deliverable Title:	Report on Guidelines and Standards for Data Treatment software

Delivery date:	Month 18
Leading beneficiary:	1 – ILL, 2 – ESS, 3 – STFC, 4 – PSI, 6 – Juelich
Dissemination level:	Public
Status:	final
Authors:	Anders Markvardsen
	STFC, ISIS Facility, UK

Project number:	654000
Project acronym:	SINE2020
Project title:	World-class Science and Innovation with Neutrons in Europe 2020
Starting date:	1st of October 2015
Duration:	48 months
Call identifier:	H2020-INFRADEV-2014-2015
Funding scheme:	Combination of CP & CSA – Integrating Activities

Abstract

This report summarises outputs from task 10.2 of WP10. The purpose of this task is to provide guidelines and standards that should be used to exploit and interface to neutron data treatment software developed in this WP.

New, and improvements to existing, neutron data treatment software are continuously needed. The reasons for this is that the field of neutron scattering is not static; there is a continuous development of new instruments, new detectors and detector electrons, improvements in computing networks and hardware, and most topical in relation to the building of ESS.

This report, in addition to an introduction, has three main sections:

- Based on answers from a questionnaire broad guidelines are derived for how to best develop software for neutron data treatment, where the questionnaire targets existing well-established data treatment software and look for commonalities between these.
- For the specific use case of writing data loaders for complex neutron data formats, what best advices, i.e. guidelines, can be given for documenting such loaders?
- Many data treatment software depends critically on fitting. This report presents a new benchmarking library for comparing fit minimizers; a required step towards providing a standard for comparing the performance and stability of minimizers used for treating/analysing neutron data.

The proposed guidelines derived from the questionnaire were discussed in detail in a presentation at the SINE2020 workshop at ILL April 2017, where many participants provided inputs. The guidelines presented take into accounts these comments and these had the broad agreement of the 58 participants of this meeting.

Table of contents

1 Introduction 4
1.1 What are guidelines and standards 4
1.2 Related publications 4
2 Guidelines based on identifying commonality from a software questionnaire 5
2.1 Software setup, communication & support 6
2.2 Software architecture 6
2.3 Software development 7
2.4 Software quality 8
2.5 In summary guidelines based on questionnaire 8
2.6 How do guidelines improve software interoperability 9
2.7 Additional guidelines suggested at SINE2020 workshop April 2017 10
3 Guidelines for documenting data loaders 10
4 Benchmarking tool for fit minimizers 12
5 Acknowledgements 14
6 Tables summarising results from the questionnaire 15
7 Appendix: In detail, results from software questionnaire 19
7.1 Reply to questionnaire: BornAgain 19
7.2 Reply to questionnaire: Mantid 23
7.3 Reply to questionnaire: McStas 30
7.4 Reply to questionnaire: MuhRec 35
7.5 Reply to questionnaire: SasView 39

1 Introduction

Software functionality is not determined by guidelines and standards. However, following such help ensure that software is easier to maintain, easier to extend as well as encourage better interoperability and sharing of ideas within a community.

This document aims to provide guidelines for how to write new (and modify existing) neutron data treatment software.

1.1 What are guidelines and standards

We come across these terms quite often and many people frequently use these terms to be mean different things (see for example: https://www.linkedin.com/pulse/20140611162901-223517409-difference-between-guideline-procedure-standard-and-policy).

Here the Oxford Dictionaries (https://en.oxforddictionaries.com/) definition of these terms will be used:

> Guideline: A general rule, principle, or piece of advice.
> *'the organisation has issued guidelines for people working with prisoners'*
>
> Standard: A required or agreed level of quality or attainment.
> *'half of the beaches fail to comply with European standards'*

Using the definition of guidelines and standards above, this report will be providing guidelines; guidelines which could be used as input into creating a standard for neutron data treatment software separately.

1.2 Related publications

The European FP7 projects, NMI3-II (www.nmi3.eu) and PaNData (www.pan-data.eu) each produced a report not directly on guidelines and standards, but closely related to this. These two reports are:

2011: PaN-data Europe, D2.2: Common policy framework on analysis software

A PaNData deliverable on a policy framework for analysis software for European photon and neutron facilities. It is an attempt at a policy on analysis software for both photon and neutron facilities. However, it is also applicable to other domains such as control system software, database applications, and office automation tools. Perhaps one challenge of this policy is that it encompasses a large scope.

2014: NMI3-II Data Analysis Standards (WP6) Task 2: Report on solutions for developing a common software infrastructure

This report gathered information about infrastructure used by neutron/muon data analysis software, and from these it derives recommendations. This report demonstrates one challenge of providing software recommendations in general. Even though the report is only three years old, some of its recommendations have been superseded by advances in technology; For example, improvements to cloud hosting sites, such as Github, have become an attractive way to host code. Encouragingly, many of the recommendations summarised in the last section of this report on infrastructure remain valid today and coincide with guidelines presented in this report.

2 Guidelines based on identifying commonality from a software questionnaire

The purpose of this software questionnaire is to collect information about the techniques, methods and tools used to develop software in the neutron and muon community. Then from this information suggest guidelines for how existing and new software for neutron data treatment software is best developed.

Many neutron data treatment software have been developed and a number of these are no longer maintained.

For this questionnaire five actively developed neutron software packages were considered. These are:

- BornAgain (http://bornagainproject.org)
- Mantid (http://www.mantidproject.org)
- McStas (http://mcstas.org)
- MuhRec (https://www.psi.ch/niag/muhrec)
- SasView (http://www.sasview.org)

These five software are all key to SINE2020 and further sample different aspects of neutron data treatment: the data analysis of grazing incidence SANS (BornAgain), software focussed on data reduction (Mantid), instrument simulation software (McStas), neutron imaging software (MuhRec) and software for the data analysis of SANS (small-angle neutron scattering) (SasView).

The questionnaire had questions on four sub-topics:

A. Software setup, communication and support
B. Software architecture
C. Software development and software quality
D. Any additional suggestions/comments

The four tables that summarise the results from the questionnaire are found in Section 6, where sub-topic C has been split into two tables. The full results from the questionnaire are presented in the Appendix.

The information in the tables in section 6 provides insight into five successful neutron data treatment software, and is a general recommended read for any new/existing software developer or project manager in this field.

In the following four subsections key software commonalities from the tables are identified and subsection 2.5 provides a summary of the guidelines drawn from this questionnaire. The following subsection discusses the community benefits of using the summarised guidelines.

Proposed guidelines derived from the questionnaire were discussed in detail in a presentation at the SINE2020 workshop at ILL April 2017, where many participants provided inputs. The guidelines

presented in the next four sections take into accounts these comments and these had the broad agreement of the 58 participants of this meeting. Furthermore, any additional guidelines proposed, not relating to questions in the questionnaire, are captured in subsection 2.7.

2.1 Software setup, communication & support

Answers to questions on software setup, communication and support are shown in table 6.1 in section 6.

Observations from the data in table 6.1 are shown below.

Table 2.1. The first column refers to row numbers in table 6.1 from which guidelines are derived.

Row	Guideline and comments
1	As a project grows the advice is to apply a governance model.
3	Which Open Source licenced is used (or planned to be used) varies, although the use of the GNU General Public License (GPL) is most common. The derived guideline is to: use an Open Source license, such as GPL. The GPL Open Source license is restrictive, but attractive for neutron data treatment software, and for most cases the GNU project recommend using GPL licence, see https://www.gnu.org/licenses/license-recommendations.html. Freely distributed software is attractive for neutron facilities, since neither facilities nor its users are then required to pay for, organise and maintain software licences.
6	Provide user support, as a minimum through email support.
7	Provide user documentation, such as version controlled user documentation.
8	With every new release, as a minimum, email out release notes to user mailing-list. As with user documentation these are advised to be version controlled.
9	Software release cycles of a few times to once a year is common, and suggests that release cycles of this length work well with facilities' needs. Furthermore, having regular releases ensures that users know the software is still supported.
10-12	Support for multiple operation systems (OSs) is also advised. This is needed to support different facilities operating systems preferences and individual user preferences.

2.2 Software architecture

Answers to questions on software architecture are shown in table 6.2 in section 6.

Observations from the data in table 6.2 are shown below.

Table 2.2. The first column refers to row numbers in table 6.2 from which guidelines are derived.

Row	Guideline and comments

1-3	Best practices software design is advised. Specifically the use of modularisation and designs to support plug-ins is commonly used by the five software packages questioned.
4	Where software needs to support scripting, the advice is to support this through Python scripting.
5	All five software packages questioned provide GUI support, and all use or are in transition to use Qt for this. Therefore, as a general rule Qt should be used for GUI development, specifically the most recent stable version of Qt. The field of computing is not static, so new GUI libraries may become available with a maturity and level of user support that is similar to Qt. Hence, the advice for which GUI library to use will likely eventually change, and therefore this is an example where the advice given in 1 and 2 should be followed to create a modular software design where GUI and non-GUI code are not tightly coupled.
6	Design software to be able to accommodate different data formats.
7	Support NeXus/HDF5 data formats. NeXus (http://www.nexusformat.org) is an effort by an international group to define a common data exchange and archival format for neutron, X-ray and muon experiments.

2.3 Software development

Answers to questions on software development are shown in table 6.3 in section 6.

Observations from the data in table 6.3 are shown below.

Table 2.3. The first column refers to row numbers in table 6.3 from which guidelines are derived.

Row	Guideline and comments
1	The use of C++ and Python is common and the advice is to use either or a combination of these two languages for neutron data treatment software.
4-5	Code maintenance is common. It is therefore advised that this is made visible to the software project throughout its life cycle and resourced appropriately. The need for maintenance (and refactoring) is to some extent a reflection of the fact that the fields of neutron instrumentation, electronics and computing are continuously advancing.
6	Using a development methodology is advised, such as an agile methodology.
7	Use Git for version control.
8	Use a build tool, such as CMake.
9	Use installer/packaging tool that makes user installation easy.
10	Use an issue/ticketing system, such as GitHub.

12	Provide developer documentation, such as version controlled developer documentation.

2.4 Software quality

Answers to questions on software quality are shown in table 6.4 in section 6.

Observations from the data in table 6.4 are shown below.

Table 2.4. The first column refers to row numbers in table 6.4 from which guidelines are derived.

Row	Guideline and comments
1-2	Some form of automated testing used together with a continuous integration tool is advised. Jenkins is used by three of the five software packages questioned.
3	For software with a large user base, it is advised to have a user testing period prior to releases.
4-5	Use static and/or dynamic analysis tools at least occasionally.
6	Do code reviews, using a tool such as GitHub.
7	It is advised to follow a coding standard.

2.5 In summary guidelines based on questionnaire

The table below summarised all the guidelines derived from the questionnaire. For more details on these see subsections 2.1-2.4.

Table 2.5. This table summarizes the guidelines derived from the questionnaire. These were discussed in detail in a presentation at the SINE2020 workshop at ILL April 2017, where many participants provided inputs. The guidelines summarized here take into accounts these comments and these had the broad agreement of the 58 participants of this meeting.

Software setup, communication and support:	**Software architecture:**
• As a project grows the advice is to apply a governance model. • Use an Open Source license, such as GPL. • Provide user support, as a minimum through email support. • Provide version controlled user documentation. • Have regular releases, and aim for a minimum of once a year. • With every new release, as a minimum, email out release notes to user mailing-list. • Support multiple operation systems.	• Design software to be modular and to support plug-ins • Where software needs to support scripting, the preference is for Python scripting • Where the software uses a GUI, the advice is to use a recent version of Qt • Design software to be able to accommodate different data formats. • Support NeXus/HDF5 data formats.
Software development:	**Software quality:**
• Use the programming languages C++ and/or Python. • Factor into your project plan that you will be doing maintenance work on your code (~10-30%). • Using a development methodology is advised, such as an agile methodology. • Use Git for version control. • Use a build tool, such as CMake. • Use tool that makes user installation easy • Use an issue/ticketing system, such as Github. • Provide version controlled developer documentation.	• Use a tool for continuous integration and automated testing, such as Jenkins. • For software with a large user base, it is advised to have a user testing period prior to releases. • Do code reviews, using a tool such as GitHub. • Use static and/or dynamic analysis tools at least occasionally. • It is advised to follow a coding standard.

2.6 How do guidelines improve software interoperability

There are many good reasons for following best practice software guidelines such to improve maintainability, expandability and stability of individual software. However, there are also community benefits. From the example of the guidelines in the previous section these are:

1. Using an Open Source license such as a GPL is a prerequisite for being able to share code and ideas expressed in code
2. Using the same programming languages further helps the above, and likewise using the same libraries, such as Qt

3. Using the same version control software, build and ticketing tools means that a developer will find it easier to help and contribute directly to other data treatment software. This is also of benefit where a developer may temporarily or permanently move from working on one data treatment software to another
4. Using the same scripting language (Python) means scripts may utilise functionality from multiple different software packages
5. Having modularized software provides easier sharing of functionality at library/modular level

2.7 Additional guidelines suggested at SINE2020 workshop April 2017

During the presentation discussing in detail guidelines from the questionnaire at the SINE2020 workshop at ILL April 2017, additional guidelines were proposed by participants and discussed. These are listed below:

1. Get DOI or arXiv citable publication out early for users to cite.
2. Consider providing Youtube video tutorials, in particular for new (non-established) software, as a way to reach a larger audience quicker.

3 Guidelines for documenting data loaders

The minimum level of documentation which is advised to be provided for any data loader is:

1. User documentation that tells the user how to use the loaders.
2. Developer documentation, including a good level of comments in the code of the loader.

However, raw data files from neutron instruments are often complex since they may store all or some of the following information:

- Counts from bespoke detector geometries, and information about these
- Information about the sample under study and its environment
- Information about the neutron beam
- Various information about the instrument during data collection
- Other information

Facilities have gradually converted towards using the NeXus format for storing raw data information, which helps. However, the locations where information is stored in NeXus files varies, which is something the NeXus committee (http://www.nexusformat.org/) is continuing to work on.

Most data treatment software will only need to load some of this information to do its job.

Therefore and because of data complexity the advice is also to:

3. Include documentation that states what specific information is loaded and where this information is stored in the software.

This additional documentation has the following benefits:

a. Advanced user documentation. I.e. for users who want to know exactly what is loaded. This can save confusion and time for developers not having to answers questions on this
b. Top-level developer documentation. Useful for experienced and novice developers to the software and needing to improve an existing loader and/or write a new similar loader. Useful for a novice developer of the software to better understand what it takes to support a new raw data file format. This saves time for any developer who wants the software to support a new instrument format and time for experienced developers having to explain how to write a data loader

An example of such documentation is shown below for the data loader LoadISISNexus, provided by Mantid developer Karl Palmen, ISIS and copied from http://docs.mantidproject.org/nightly/algorithms/LoadISISNexus-v2.html#algm-loadisisnexus:

Data loaded from Nexus File

Not all of the nexus file is loaded. This section tells you what is loaded and where it goes in the workspace.

The nexus file must have `raw_data_1` as its main group and contain a `/isis_vms_compat` group to be loaded.

The workspace data is loaded from `raw_data_1/Detector_1`.

Instrument information is loaded from `raw_data_1/Instrument` if available in file, otherwise instrument information is read from a MantidInstall instrument directory.

Also the `NSP1`, `UDET`, `SPEC`, `HDR`, `IRPB`, `RRPB`, `SPB` and `RSPB` sections of `raw_data_1/isis_vms_compat` are read. The contents of `isis_vms_compat` are a legacy from an older ISIS format.

Here are some tables that show it in more detail:

Description of Data	Found in Nexus file (within 'raw_data_1')	Placed in Workspace (Workspace2D)
Monitor Data	within groups of Class NXMonitor (one monitor per group)	Monitor histogram data (loaded depending on prop. LoadMonitors)
Detector Data	group `Detector_1` (all detectors in one group)	Histogram Data
Instrument	group `Instrument`	Workspace instrument
Spectrum of each detector ID	`NSP1`, `UDET` and `SPEC` within `isis_vms_compat`	Spectra-Detector mapping
Run	various places as shown below	Run object
Sample	`SPB` and `RSPB` within `isis_vms_compat`	Sample Object

Run Object

LoadISISNexus executes LoadNexusLogs v1 to load run logs from the Nexus `runlog` or some other appropriate group. It also loads the Nexus `raw_data_1/periods/proton_charge` group into the `proton_charge_by_period` property of the workspace run object.

Properties of the workspace Run object are loaded as follows:

Nexus	Workspace run object
HDR	run_header (spaces inserted between fields)
title	run_title
start_time	run_start
end_time	run_end
(data)	nspectra
(data)	nchannels
(data)	nperiods
IRPB[0]	dur
IRPB[1]	durunits
IRPB[2]	dur_freq
IRPB[3]	dmp
IRPB[4]	dmp_units
IRPB[5]	dmp_freq
IRPB[6]	freq
IRPB[7]	gd_prtn_chrg
RRPB[9]	goodfrm
RRPB[10]	rawfrm
RRPB[11]	dur_wanted
RRPB[12]	dur_secs
RRPB[13]	mon_sum1
RRPB[14]	mon_sum2
RRPB[15]	mon_sum3
RRPB[21]	rb_proposal

In the Nexus column, names of groups in capitals are in raw_data_1/isis_vms_compat and the other groups are in raw_data_1.

(data) indicates that the number is got from the histogram data in an appropiate manner.

IRPB and RRPB point to the same data. In IRPB, the data is seen as 32-bit integer and in RRPB it is seen as 32-bit floating point (same 32 bits). In all cases, integers are passed. The 32-bit floating point numbers are used only to store larger integers.

The other indices of IRPB and RRPB are not read.

Sample Object

Properties of the workspace sample object are loaded as follows:

Nexus	Workspace sample object
SPB[2]	Geometry flag
RSPB[3]	Thickness
RSPB[4]	Height
RSPB[5]	Width

Nexus groups are found in raw_data_1/isis_vms_compat. Other indices of SPB & RSPB are not read.

4 Benchmarking tool for fit minimizers

Fitting is a core functionality in many neutron and muon data treatment software packages, including most of those worked on as part of this WP.

Within this WP a first version of a new tool/library for comparing fit minimizers targeted for the fitting of neutron data has been created, and most of the code development work for this was done by Federico Montesino Pouzols.

When fitting a function to experimental or simulated data, the function parameters are chosen so that the model fits the data as closely as possible. A measure of how close a fit is to the data is defined by a cost function, and the smaller the cost function value the better the model fit the data.

Many minimization methods (minimizers) exists, and currently there does not exist an agreed standard method for objectively compare minimizers used in neutron data treatment software. Perhaps this explains why different neutron data treatment software authors favour different minimizers.

Work has been done as part of this WP to create a new tool for comparing fit minimizers. The main categories of minimizers relevant to neutron data treatment are:

1. Local minimizers: From an initial guess of the fit function parameter values, the minimizer will search for the local minimum.
2. Global minimizers: Here, in general, you don't have a clue what the parameters are and want the global minimizer to tell you which parameter values are best – globally.
3. Sampling 'minimizers': You want to sample effectively an area of parameter space, for example, for the purpose of mapping out Bayesian posterior probabilities distributions. Perhaps the word 'minimizer' is not perfect in this case, but nevertheless commonly used for this.

Other related optimisation problems are also relevant for some neutron data treatments including heavily constrained problems and problems aiming to identify other features such as saddle points.

A tool for comparing local minimizers was targeted (in line with available resources, deadlines and strongest need). Local minimizers are relevant to many software, including Mantid where this account for the vast majority of the fitting done with this software.

A tool has been created, which simply consists of a library of fit problems and a few Python files (currently located https://github.com/mantidproject/mantid/tree/master/scripts/CompareFitMinimizers) that loops over the fit problems using different minimizers, where the current set of fit problems targets the testing of local minimizers.

Work done using this tool has been reported in an SINE2020 news report http://sine2020.eu/news-and-media/improving-the-fitting-in-mantid-for-neutron-and-muon-data.html, where this tool is used to compare existing minimizers with a newly developed minimizer by the Computational Mathematics Group of the STFC Scientific Computing Department, UK named Trust Region.

This tool is now daily system-tested in Mantid and it is manually run before each official release to check if any new improvements have affected the overall performance of the minimizers. Based on the latter a summary description of which minimizers to use with Mantid is updated accordingly, see the top section of the URL: http://docs.mantidproject.org/nightly/concepts/FittingMinimizers.html.

This tool has been designed so that it can be used to compare mininizers not just those implemented in Mantid, but any callable minimizer from Python. For an expert Python developer this can be achieved by modifying the existing Python files in https://github.com/mantidproject/mantid/tree/master/scripts/CompareFitMinimizers. Furthermore, work is currently in progress to make this easier to do by a non-expert Python developer.

5 Acknowledgements

Thanks to Catherine Jones, STFC, UK for early discussions on writing standards and guidelines. Thanks to Joachim Wuttke, Wojtek Potrzebowski, Anders Kaestner, Peter Willendrup and Nick Draper for filling in the questionnaire. Thanks to Thomas Holm Rod for overall help with this report, Inês Crespo for writing a news article on the fit benchmarking tool and Branwen Hide for help with SINE2020 financial questions. Thanks to Nick Gould, Jennifer Scott, Tyrone Rees and Roman Tolchenov for inputs into the fit benchmarking tool. Thanks to Steve Cottrell and Anthony Lim for proof reading and comments to this report, and to participants of the SINE2020 workshop at ILL April for valuable comments to the content in this during a presentation.

6 Tables summarising results from the questionnaire

The next four tables summarise the data from the questionnaire detailed in the Appendix.

An empty field in a table means that no answer was provided to that question.

Table 6.1. Software setup, communication & support. The 1st column shows the row number.

		BornAgain	Mantid	McStas	MuhRec	SasView
1	**Governance model**	None	Project Management Board comprised of members of the contributing facilities	Informal. Notion of core group	Advisory committee	The management team consists of 4 people from different neutron facilities
2	**Age of project**	Started 2012	~9 years	~19 years	~8 years	Started 2006
3	**License**	GPL3	GPL3	GPL (2.0)	Freeware (1)	BSD
4	**# of developers**	3	~20 split across 4 sites	~5	2	2 full time, approx. 20 occasionally
5	**# of location software developed**	Single	Multiple	Multiple	Single	Multiple
6	**User support**	Through mail and Web contact form	Via a forum, email and phone	User mailing list and emails send directly to main authors	By direct email communication	Through the user mailing list and tutorial
7	**User documentation**	Installation instructions, tutorials and examples online. Physics manual (pdf) still incomplete	Web-based and built-in docs, training materials that can be followed in your own time and which is used in training courses. (2)	PDF's, html online docs, wikis. We are considering to create webcasts for installation and introductory use. (3)	Muhrec has a user manual. No learning material is provided	Web-based and built-in documentation and a standalone tutorial
8	**How is a release communicated to users**		Release notes emailed to user mailing list, release presentations and annual scientific steering committee meeting	User mailing list	Currently none	E-mail to mailing list with release notes
9	**Release cycles**	A few releases per year	~ every 4 months	~ 1 per year	About once a year	~ 1 per year
10	**Windows support**	Yes	yes	yes	Yes	yes
11	**Mac support**	yes	yes	yes	Yes	yes
12	**Linux support**	yes	yes	yes	Yes	partially

1 Is being moved to Open Source

2 And Mantid-Muon introductory material through https://www.e-neutrons.org

3 And McStas simulation taster material through https://www.e-neutrons.org

Table 6.2. Software architecture. The 1st column shows the row number.

		BornAgain	Mantid	McStas	MuhRec	SasView
1	**Architecture principle**	Standard modern C++ design	Separation of Data and Algorithms. Encapsulated "User Code". Reuse of existing components and careful memory management when handling large datasets	"Less is more", simplicity and readability over speed. Limit number of external dependencies	The architecture is object oriented and generalization allows to separate GUI from core code	The software is divided into: calculation module, model function library and GUI. There is also a minimizer module that is separately distributed
2	**Modularisation**	Core library for modelling sample and instrument, and computing the expected detector image. Python wrapper and GUI on top of it	The framework is split into several modules, as is the user interface, and both are separate from the other	Layers: GUI tools, "core" code generator, instrument file, component codes, simulation binaries	The project is divided into core code and GUI code and each part is subdivided into dedicated libraries	Sasmodels is a separate library (separately hosted on Github) that can be interfaced with the minimization module directly. Calculation and UI modules are also independent
3	**Support plug-ins**	No plug-in mechanism at C++ level, but the Python wrapper allows users to implement custom form factors	Plug-in architecture that allows users to extend the framework in C++ and Python by defining their own algorithms, custom user interfaces or data objects, among many other things	Components are our plug-ins. Written as ISO C code in a structured format	Yes, the tools support dynamic loading of precompiled share library objects	Model functions can be supplied as plug-ins
4	**Is scripting supported**	Python scripting	Python scripting	Scripting is provided by the tool layer	Scripting is supported through CLI arguments	Python scripting and internal scripting in plugin editor
5	**GUI framework**	Qt-based	Qt-based	Moving to Python-Qt, Perl-Tk is our legacy solution	Qt-based	Moving from wx-Python to Qt
6	**Can accommodate different data formats**	yes	Yes, Mantid has a list of formats that it can write to or read from, and this is extensible a plug-in mechanism	Yes: We provide our own "McStas" ascii format as well as NeXus output	Currently, fits, tiff, and nexus are supported. Adding new formats must be done on the code level	yes
7	**Any data formats specifically supported for interoperability with other software**	Import and export formats are provided as users request this	Yes, apart from one internal format, all of the other formats we support were created to allow interaction with other software	For ease of use with Mantid, our NeXus output can embed an XML based IDF	The chosen formats are the most commonly used in neutron imaging	Reads and writes CanSAS 1D xml and NXCansas HDF5. Supports input and output in basic multi-column text files.
8	**API for external software provided**	yes	A simple C-style API as well as much more extensive C++ and Python APIs	No formal API defined. The instrument file format is however well described and the cmdline interfaces simple	The modularity of the core libraries make reusing of the code possible, but is not on specific intention	No formal API yet but documentation efforts to encourage reuse in other python based software or in Jupyter notebooks are under way.

Table 6.3. Software development. The 1st column shows the row number.

		BornAgain	Mantid	McStas	MuhRec	SasView
1	**Code languages**	C++ and Python	C++ and Python	C, Python and Perl (Perl being replaced with Python)	C++ and Python	C++ and Python
2	**Compilers supported**	GCC, waiting for Clang to fully work	MSVC, GCC and Clang	"Anything ISO-c"	Windows: MSVC2015, MacOS: XCode, Linux: g++.	OSX – clang, GCC. Windows – msvc, tinycc
3	**Bindings**	SWIG binding C++ -> Python	Boost.Python and SIP	Perl-PGPLOT, Perl-Tk, Python-Qt etc.		No
4	**~% of development goes into maintenance**	10-20%	2 weeks after each release is devoted to code maintenance. Many other tasks can be classified as maintenance	10-30%	10%	20-30%
5	**How frequently is code re-engineer**	Continually; almost every new functionality requires some refactoring	Extensive or risky changes are reserved for the maintenance periods	Often	Rarely	Not too often. Rather bug fixing
6	**Development methodology**	Agile	Agile approach that has been adjusted for the distributed nature of the development team	Generally informal/undefined. On some ESS-oriented tasks, the methodology is SCRUM	Waterfall	Agile development with scrum and 2-3 week long sprints implemented internally at ESS
7	**Version control**	Git	Git	Git	SVN but transfer to Git planned in 2017	Git
8	**Build and configuration tools**	CMake	CMake	CMake	QMake	Python packaging: setuptools, distutils. Make for building documentation of model function library (sasmodels)
9	**Packaging and installers**	Linux: source tgz. MacOS: dmg. Windows: installer exe	We use CPack to create our installers	CPack plus home-grown "metapackages"	Currently, the packaging is done by shell scripts	OSX – py2ap, Windows – pyinstaller
10	**Issue/ticket tracking system**	Redmine, soon moving to GitHub	GitHub	GitHub	GithHub	Trac
11	**Organized events**	Not yet	We hold an annual developer workshop, where external contributors are invited also	Every ~6-12 months to work on common tasks: "code camps". User workshops most years, 3 in 2016	No. The team currently sit in the same office	Code camps – once or twice a year
12	**Developer documentation**	wiki pages; Doxygen comments and other comments in the sources	In developer wiki	In developer wiki	Doxygen generated API documentation. Some few framework module guide documents	Little and rather scattered

Table 6.4. Software quality. The 1st column shows the row number.

		BornAgain	Mantid	McStas	MuhRec	SasView
1	**Automated testing**	For unit tests, functional tests, and integration tests	For unit testing, system testing and documentation testing. We are moving many of our UIs to an MVP pattern to allow for automated testing of UI logic	Yes: but not in a very strict way. Every night an integration test runs – based on a nightly build	Unit testing is used, but with low coverage	Unit tests to a limited extent but are planning to do more
2	**Continuous integration**	buildbot	Jenkins, using Leeroy to run build for all of our Pull requests	Currently a home-grown script-oriented system running on OSX, Debian, Windows. We are moving to Jenkins	First build scripts are implemented with Jenkins	Build servers for Windows, OSX and partialy for Linux. Jenkins based
3	**Acceptance testing**	no	For each release we allow for 2 weeks of beta testing	Yes, but not in a formalised sense	No	Yes but only before releases
4	**Static code analysis tools**	occasionally	Several, Clang format, cppcheck, coverity, pylint, clang tidy	On occasion yes, formalised no	No	Pylint, Quantified code
5	**Dynamic analysis tools**	occasionally Valgrind	Valgrind, AddressSanitizer, very sleepy	On occasion yes, formalised no	Intel Profiler long time ago	no
6	**Do you do code reviews**	Under consideration, as we move to GutHub	All changes to the code are reviewed by a different developer as part of the testing process	Code reviews at DTU between developers there – code, demo and comments in a weekly meeting	No	Through GitHub pull requests
7	**Follow coding standards**	Our own standards, slowly evolving, documented in internal wiki	Standards for C++, Python, units tests and Mantid algorithms and fitting functions	Not in formalised way	No, clean-up is needed	The coding standards are in preparation and are mostly collection of PEP guidelines and some SasView specific rules

7 Appendix: In detail, results from software questionnaire

The questionnaire was completed for the following the data treatment software:

- BornAgain (http://bornagainproject.org)
- Mantid (http://www.mantidproject.org)
- McStas (http://mcstas.org)
- MuhRec (https://www.psi.ch/niag/muhrec)
- SasView (http://www.sasview.org)

It contains four sections:

E. Software setup (+ communication and support)
F. Software architecture
G. Software development and software quality
H. Any additional suggestions/comments

7.1 Reply to questionnaire: BornAgain

Answer to the questionnaire for the BornAgain software.

A. Software setup

A.1. Overview

What is the audience? What is the user base?

Users of GISAXS and GISANS instruments.

Who contributes? Who can contribute?

Currently 3 full-time developers in Scientific Computing Group at MLZ Garching. Some users contribute valuable feedback. Everybody is welcome to contribute comments or patches.

Governance model used (e.g. does it include a management board)

None.

Number of developers working on the software

3, see above.

Is the development team in a single or multiple locations?

Single location so far. This may change next year with a co-developer being hired by University of Erlangen.

License of your software

GPL v3 or higher

License(s) of third party libraries and software required to use the software. Are they more restrictive?

GPL or less restrictive.

Is user support available? What means of support?

Yes, through mail (contact@bornagainproject.org) or Web contact form (http://bornagainproject.org/contact)

Post-release communication with users (is there any form of stakeholder engagement?)

?

Age of the project

started 2012, first release 2014

A.2. Requirements. How are software requirements gathered.

Who has a say in defining requirements?

Initial requirements defined by institute management in a proposal for the Helmholtz Gemeinschaft. Additional requirements collected from user feedback and from in-house research needs.

Do you collate requirements? How are all the requirements collated?

In our issue tracker.

Do you define priorities? Deadlines?

Priorities defined at start of each "sprint". No hard deadlines.

B. Software architecture

B.1. Architecture

What are the general architectural principles?

Standard modern C++ design.

Modularization. Are there modules, independent libraries, separate layers?

Core library for modeling sample and instrument, and computing the expected detector image. Python wrapper and GUI on top of it.

Does it support plug-ins (for example for fitting functions and minimizers)?

No plug-in mechanism at C++ level, but the Python wrapper allows users to implement custom

form factors.

Is the system extensible? Is scripting supported?

Yes, Python scripting.

B.2. User interfaces

Do you provide a graphical user interface (GUI)? On what platform is it supported? What framework is it based on?

Yes. Multi-platform with active support for Linux, Mac, Windows. Qt-based.

Do you provide a command line interface (CLI)?

No explicit support, but comes automatically with Python wrapper.

B.3. Interoperability

Does the software interoperate with other software? How? For example, via system calls, scripting, file I/O, web services, command line?

We use generic software like matplotlib. No interoperation with other domain-specific software.

Is it possible to accommodate different data formats?

Yes.

Are any data formats supported specifically for interoperability with some third party software?

Import and export formats are provided as users request.

Does it have an API that can be used from external software?

Yes.

Where do you see opportunities for and advantages of interoperability?

Right now not under consideration.

C. Software development and software quality

Development methodology used (Agile/Waterfall/other). Please provide comments. If you use agile techniques for example, which ones

Agile, somehow.

Are version control tools used and how? What tool(s) (e.g. Git, Subversion, others)?

git.

Do you use any issue/ticket tracking system? What tools do you use, e.g. Trac, Jira, Redmine, or GitHub issues? How do you use it?

Redmine, soon moving to GitHub

Do you use automated testing (e.g. unit, integration, unit, system, testing?

Yes, fully automatized unit tests, functional tests, and integration tests.

Do you use any form of acceptance testing (e.g. beta or alpha testing)?

No.

Do you have a release cycle?

A few releases per year, no fixed dates.

Do you do code reviews? What tools? How often and/or how much percentage of the code?

Under consideration, as we move to GutHub.

Do you organize events such as code camps, hackathons, developer workshops, etc.? Please describe them.

Not yet.

Is continuous integration used and how (build servers)? What tools are used, for example Jenkins, Travis, etc.?

Yes. buildbot.

What build and configuration tools are used? For example CMake, pkg-config, make, ninja, auto-tools, Python packaging systems, etc.

CMake.

What operating systems/platforms are supported? Are all of them fully supported?

Linux, MacOS, MS Windows all fully supported.

Packaging and installers. What packages do you distribute? How are they generated? Do you use any specific tool such as CPack, custom package templates, etc.?

Linux: source tgz. MacOS: dmg. Windows: installer exe.

Compilers supported

gcc, waiting for Clang to fully work

Do you use static code analysis tools (e.g. Clang tools, cpplint, Eclipse, pylint, pep8)?

occasionally

Do you use dynamic analysis tools for debugging and/or profiling? Examples: Valgrind, Clang tools such as AddressSanitizer.

occasionally Valgrind

User documentation. Do you provide for: installation, user manual, tutorials, user workshops, other learning materials (videos, are specific guides, etc.)?

Installation instructions, tutorials and examples online. Physics manual (pdf) still incomplete.

What developer documentation is available?

Some wiki pages; Doxygen comments and other comments in the sources.

What programming languages are used and in what proportion?

C++. Automatically generated Python wrapper. Little additional Python. Large collection of Python examples.

Are there bindings between programming languages?

Yes, SWIG binding C++ -> Python.

Do you follow and/or enforce any programming or coding standards (e.g. Google C++ Style Guide)?

Our own standards, slowly evolving, documented in internal wiki.

Do you put effort in code maintenance. What approximately percentage of development goes into maintenance?

10-20%, perhaps.

How frequently do you reengineer or refactor code?

Continually; almost every new functionality requires some refactoring.

7.2 Reply to questionnaire: Mantid

Answer to the questionnaire for the Mantid software.

A. Software setup

A.1. Overview

What is the audience? What is the user base?

Anyone working with neutron scattering and muon data, including visiting scientists as well as facility staff, with an emphasis on users who want to manipulate and visualize such data.
Who contributes? Who can contribute?
The main development team is a collaboration between ISIS, STFC the SNS, ORNL, the ESS and the ILL. Other people can contribute through the GitHub pull request mechanism.
Governance model used (e.g. does it include a management board)
Mantid has a Project Management Board comprised of members of the contributing facilities.
Number of developers working on the software
The development team total to over 20 FTE's of developers, split across the 4 main contributing facilities.
Is the development team in a single or multiple locations?
The development team is split across ISIS, the SNS, the ESS and ILL facilities.
License of your software
Mantid is released under GPL v3 or higher.
License(s) of third party libraries and software required to use the software. Are they more restrictive?
None of the third party libraries used at present are more restrictive, in fact many are less so. The third party libraries are licensed under a mixture of LGPL, GPL, Berkley and custom open source licenses.
Is user support available? What means of support?
User support is available via our forum, help.mantidproject.org, and via email and phone within any of the contributing facilities. Users or staff at one of the contributing facilities will have preferential access to support resources above external users.
Post-release communication with users (is there any form of stakeholder engagement?)
We hold regular training sessions at ISIS and the SNS, annual Mantid scientific steering committee is held at one of the facilities to gather future requirements and directions.
Age of the project
9 year

A.2. Requirements. How are software requirements gathered.

Who has a say in defining requirements?

Requirements can come from many sources: The scientific steering committee, Facility IT and scientific strategies, Instrument scientists, the development team and suggestions from external users.

Do you collate requirements? How are all the requirements collated?

Requirements were initially collated in the User Requirements Document at the start of the project, but are now stored as Github issues in the Mantid GitHub project.

Do you define priorities? Deadlines?

Priorities are defined by a simple low, medium, high scale together with assigning issues to a particular planned release to fit in with any defined deadline.

B. Software architecture

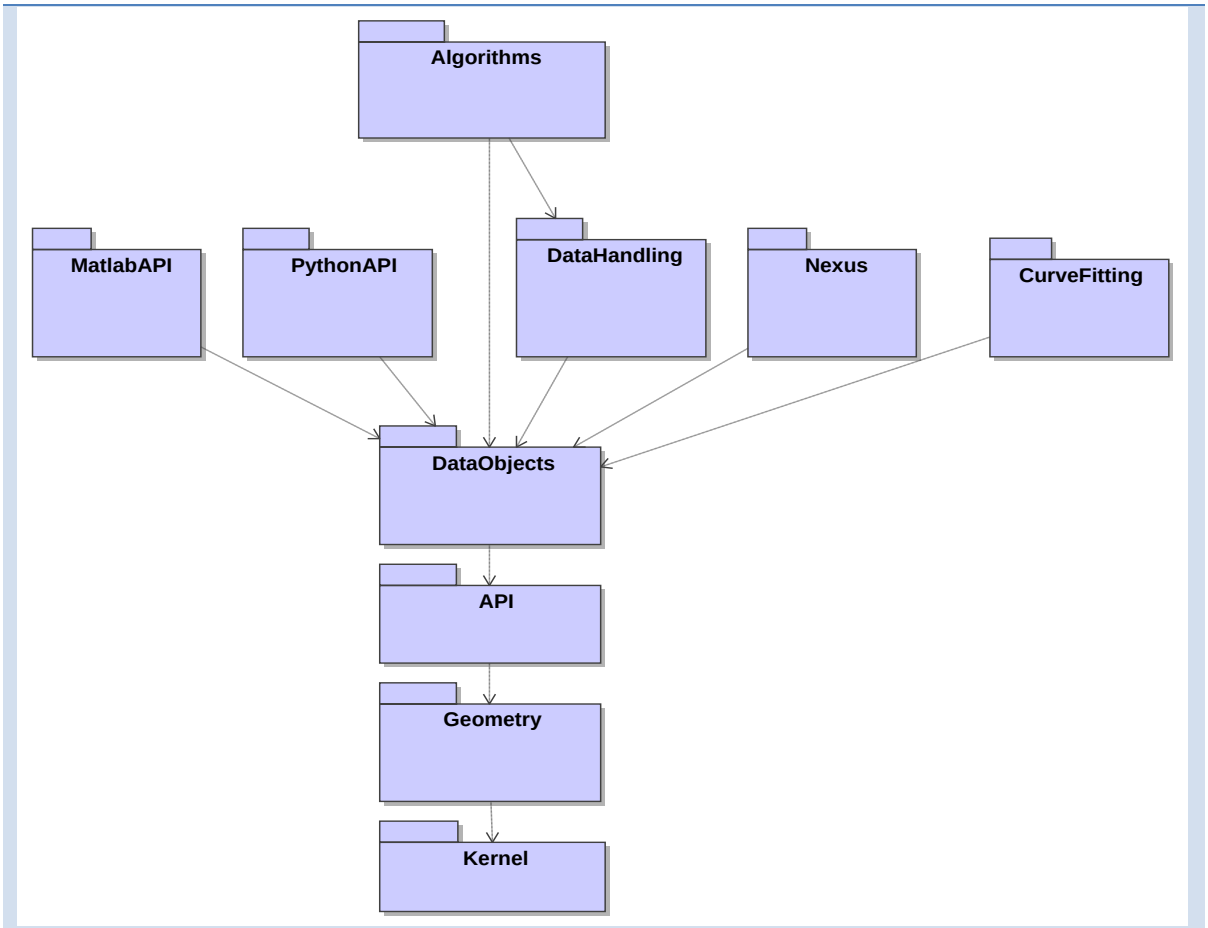
B.1. Architecture

What are the general architectural principles?

The goal is "Consolidate the data reduction/analysis software for neutron scattering without restricting the needs of the instrument scientists". The design principles included: Separation of Data and Algorithms, Encapsulated "User Code" in specific places, use of well designed interfaces to allow generic use of components, Reuse of existing components where possible and careful memory management when handling large datasets.

Modularization. Are there modules, independent libraries, separate layers?

Yes The framework is split into several modules, as is the user interface, and both are separate from the other. Below is a simplified package diagram of the framework.



Does it support plug-ins (for example for fitting functions and minimizers)?

The Mantid framework uses a plug in architecture that will allow users to extend the framework by defining their own algorithms, custom user interfaces, data objects or services, among many other things.

Is the system extensible? Is scripting supported?

Yes the system is extensible through both C++ and python. We hold training courses in using python to extend the framework for both algorithms and fitting functions.

B.2. User interfaces

Do you provide a graphical user interface (GUI)? On what platform is it supported? What framework is it based on?

Mantid supports several graphical user interfaces, some hosted within the generic Mantidplot interfaces as well as other external tools. All of the tools hosted within the Mantidplot interface are supported on Windows, Linux (Ubuntu and RHEL) and Mac OsX. SNS and ISIS also have web interfaces to their auto reduction facilities that use the Mantid Framework to process the data.

Do you provide a command line interface (CLI)?

Mantid can be used without a user interface through Python. Otherwise Python scripts can also be executed through the Mantidplot command line interface.

B.3. Interoperability

Does the software interoperate with other software? How? For example, via system calls, scripting, file I/O, web services, command line?

Within an algorithm inside Mantid the developer has access to any other software they wish to use, so this question is hard to answer. However at present we use all of the listed approaches for interacting with third party software.

Is it possible to accommodate different data formats?

Mantid has an extensive list of formats that it can write to or read from, and this is extensible suing our plug in algorithm system.

Are any data formats supported specifically for interoperability with some third party software?

Yes, apart from one internal format, all of the other formats we support were created to allow interaction with other software.

Does it have an API that can be used from external software?

Yes, Manti has a simple C style API as well as much more extensive C++ and Python APIs.

Where do you see opportunities for and advantages of interoperability?

Mantid always looks for opportunities to interact with existing tools and software. If an existing tool or package exists that is an accepted tool or standard for a task we would look to interact with that tool rather than reimplement functionality.

C. Software development and software quality

Development methodology used (Agile/Waterfall/other). Please provide comments. If you use agile techniques for example, which ones

Mantid follows an Agile approach that has been adjusted for the distributed nature of the development team, the multiple facility wide base of key users. Key differences are the long iterations and releases that fit with an acceptable release schedule for our facilities, and the use of named developer – scientist pairings to provide the product owner role for each technique.

Are version control tools used and how? What tool(s) (e.g. Git, Subversion, others)?

We use Git, hosted in Github.

Do you use any issue/ticket tracking system? What tools do you use, e.g. Trac, Jira, Redmine, or GitHub issues? How do you use it?

We use Github Issues and pull requests. Previously we used Trac, but found the performance at remote sites unacceptable.

Do you use automated testing (e.g. unit, integration, unit, system, testing?

We use automated testing extensively, for unit testing, system testing and documentation testing. We are moving many of our UIs to an MVP pattern to allow for automated testing of a lot of the UI logic.

Do you use any form of acceptance testing (e.g. beta or alpha testing)?

For each release we allow for 2 week of beta testing.

Do you have a release cycle?

Yes we have a release cycle that has been agreed with the contributing facilities through the PMB. Mantid releases roughly every 4 months.

Do you do code reviews? What tools? How often and/or how much percentage of the code?

All changes to the code are reviewed by a different developer as part of the testing process before that development branch can be merged into the master code branch. We have a gatekeeper group of senior developers that ensure this process happens.

Do you organize events such as code camps, hackathons, developer workshops, etc.? Please describe them.

We hold an annual developer workshop, often alongside our annual scientific steering committee meetings. The development team and external contributors are invited.

Is continuous integration used and how (build servers)? What tools are used, for example Jenkins, Travis, etc.?

We use Jenkins to provide continuous integration, using Leeroy to run build for all of our Pull requests.

What build and configuration tools are used? For example CMake, pkg-config, make, ninja, auto-tools, Python packaging systems, etc.

CMAKE, some developers use ninja underneath to improve build times.

What operating systems/platforms are supported? Are all of them fully supported?

http://www.mantidproject.org/Supported_Operating_Systems

Packaging and installers. What packages do you distribute? How are they generated? Do you use any specific tool such as CPack, custom package templates, etc.?

We use CPack to create our installers.

Compilers supported

Internally we use MSVC, GCC and Clang

Do you use static code analysis tools (e.g. Clang tools, cpplint, Eclipse, pylint, pep8)?

Several, Clang format, cppcheck, coverity, pylint, clang tidy

Do you use dynamic analysis tools for debugging and/or profiling? Examples: Valgrind, Clang tools such as AddressSanitizer.

Valgrind, AddressSanitizer, very sleepy

User documentation. Do you provide for: installation, user manual, tutorials, user workshops, other learning materials (videos, are specific guides, etc.)?

Yes, we have extensive documentation http://docs.mantidproject.org that are also installed with each release, we also have training materials that can be followed in your own time or are presented as training courses twice per year.

What developer documentation is available?

http://www.mantidproject.org/Category:Development

What programming languages are used and in what proportion?

The majority of the code is in C++ and Python. Proportions are here https://www.openhub.net/p/Mantid/analyses/latest/languages_summary

Are there bindings between programming languages?

Mantid has extensive Python Bindings

Do you follow and/or enforce any programming or coding standards (e.g. Google C++ Style Guide)?

Mantid follows it's own coding standards
http://www.mantidproject.org/Coding_Standards

Do you put effort in code maintenance. What approximately percentage of development goes into maintenance?

We have a minimum of 2 weeks after each release devoted to code maintenance, but many other tasks could be classified as maintenance as well.

How frequently do you reengineer or refactor code?

Whenever it is necessary. Extensive or risky changes are reserved for the maintenance periods.

7.3 Reply to questionnaire: McStas

Answer to the questionnaire for the McStas software.

A. Software setup

A.1. Overview

What is the audience? What is the user base?

Instrument designers, instrument scientists, experimentalists.

Who contributes? Who can contribute?

Contributions are typically by instrument scientists or university users. In principle anyone with a GitHub account can contribute.

Governance model used (e.g. does it include a management board)

Very informal. We have the notion of a "core group" with a corresponding mailing list.

Number of developers working on the software

Currently ~ 5

Is the development team in a single or multiple locations?

Multiple locations

License of your software

GPL (2.0)

License(s) of third party libraries and software required to use the software. Are they more restrictive?

GPL or other compatible license preferred. Some licenses are less restrictive.

Is user support available? What means of support?

Users can pose questions to the user mailing list and some send mails to the main authors directly. More intensive local support is offered at some facilities by main authors, but also from local "superusers".

Post-release communication with users (is there any form of stakeholder engagement?)

User mailinglist

Age of the project

18-19 years, founded in 1997 with first release in 1998

A.2. Requirements. How are software requirements gathered.

Who has a say in defining requirements?

The "core" team

Do you collate requirements? How are all the requirements collated?

We try to gather requirements, requests etc. as GitHub issues. The list is however often incomplete.

Do you define priorities? Deadlines?

Priorities are defined pr. GitHub issue. Deadlines are generally very soft.

B. Software architecture

B.1. Architecture

What are the general architectural principles?

"Less is more", simplicity and readability over speed. Limit number of external dependencies.

Modularization. Are there modules, independent libraries, separate layers?

Layers: GUI tools, "core" code generator, instrument file, component codes, simulation binaries

Does it support plug-ins (for example for fitting functions and minimizers)?

Components are our plug-ins. Written as ISO C code in a structured format.

Is the system extensible? Is scripting supported?

Yes, for instance via our component. We recently initiated support for providing external (c / fortran lib) dependencies. Scripting is provided by the tool layer (simulation series, optimisations)

B.2. User interfaces

Do you provide a graphical user interface (GUI)? On what platform is it supported? What framework is it based on?

Linux, Mac OS, Windows. We are actively moving to Python-Qt, Perl-Tk is our legacy solution.

Do you provide a command line interface (CLI)?

Yes. Generally experienced users end up running by use of favourite text editor and the command line interface.

B.3. Interoperability

Does the software interoperate with other software? How? For example, via system calls, scripting, file I/O, web services, command line?

Yes. Examples of all of the above exist.

Is it possible to accommodate different data formats?

Yes: We provide our own "McStas" ascii format as well as NeXus output.

Are any data formats supported specifically for interoperability with some third party software?

For ease of use with Mantid, our NeXus output can embed an XML based IDF.

Does it have an API that can be used from external software?

No formal API defined. The instrument file format is however well described and the cmdline interfaces simple.

Where do you see opportunities for and advantages of interoperability?

Recruitment route from other softwares, welcoming users from other areas. New areas of use/application.

C. Software development and software quality

Development methodology used (Agile/Waterfall/other). Please provide comments. If you use agile techniques for example, which ones

Generally informal/undefined. On some ESS-oriented tasks, the methodology is SCRUM.

Are version control tools used and how? What tool(s) (e.g. Git, Subversion, others)?

Yes. Full revision history exist. Currently Git (on GitHub), was on lab-driven SVN, CVS, RCS before this. Also see https://www.openhub.net/p/mcstas. Repos shared with McXtrace (our sister-code for X-rays.)

Do you use any issue/ticket tracking system? What tools do you use, e.g. Trac, Jira, Redmine, or GitHub issues? How do you use it?

GitHub issues, was Trac earlier end originally Bugzilla. No formal policy on the use, but we try to put new features, issues, user bugs on there. Shared with McXtrace

Do you use automated testing (e.g. unit, integration, unit, system, testing?

Yes, but not in a very strict way. Every night an integration test runs – based on a nightly build. Output available at http://nightly.mccode.org - including latest results at http://nightly.mccode.org/0_Current.html and graphical test output over time at

http://nightly.mccode.org/Datafiles/ . Shared with McXtrace

Do you use any form of acceptance testing (e.g. beta or alpha testing)?

Yes, but not in a formalised sense.

Do you have a release cycle?

Effectively yes - ~ 1 release / Y. But not formalised. We are trying to move to a more agile scheme.

Do you do code reviews? What tools? How often and/or how much percentage of the code?

Code reviews at DTU between developers there – code, demo and comments in a weekly meeting. No software tools.

Do you organize events such as code camps, hackathons, developer workshops, etc.? Please describe them.

Informally yes. We try to meet up every 6-12 months and work on common tasks. "code camps". User workshops most years, 3 in 2016.

Is continuous integration used and how (build servers)? What tools are used, for example Jenkins, Travis, etc.?

Currently a home-grown script-oriented system running on OSX, Debian, Windows. We are moving to Jenkins.

What build and configuration tools are used? For example CMake, pkg-config, make, ninja, auto-tools, Python packaging systems, etc.

CMake is used (we came from auto-tools). For Python packages we are relying on Anaconda

What operating systems/platforms are supported? Are all of them fully supported?

Linux (Debian-based preferred), Mac OS X (~3 most current releases, others available on request), Windows (32 bit packages that work on 64 bit). Windows solution is in some areas handicapped as it does not work exactly the same. All functionality is however available everywhere.

Packaging and installers. What packages do you distribute? How are they generated? Do you use any specific tool such as CPack, custom package templates, etc.?

For Debian: CPack plus home-grown "metapackages" (equivs-build). For RPM's CPack plus home-grown "metapackages" (rpmbuild). For OSX: CPack plus home-grown "metapackages" (PackageMaker). For Windows: CPack (cross-compile with mingw on Linux) plus home-grown "metapackage" (inno-setup running on Linux)

Compilers supported

"Anything ISO-c" (effectively people build with GNU, Intel C or clang/llvm-gcc)

Do you use static code analysis tools (e.g. Clang tools, cpplint, Eclipse, pylint, pep8)?
On occasion yes, formalised no. For code stats, see https://www.openhub.net/p/mcstas
Do you use dynamic analysis tools for debugging and/or profiling? Examples: Valgrind, Clang tools such as AddressSanitizer.
On occasion yes, formalised no. For code stats, see https://www.openhub.net/p/mcstas
User documentation. Do you provide for: installation, user manual, tutorials, user workshops, other learning materials (videos, are specific guides, etc.)?
Yes, all of the above as written material (PDF's, html online docs, wikis). We are considering to create webcasts for installation and introductory use.
What developer documentation is available?
A developer wiki is being populated at https://github.com/McStasMcXtrace/McCode/wiki
What programming languages are used and in what proportion?
Mostly c (51,2 KloC), TeX docs (18,5 KloC), tools in Perl (13,6 KloC) and Python (11,2 KloC) plus various graphical backends. For a full, updated statistics, see https://www.openhub.net/p/mcstas/analyses/latest/languages_summary
Are there bindings between programming languages?
Yes, mostly on the tool/graphics side. (perl-PGPLOT, perl-Tk, Python-Qt etc.)
Do you follow and/or enforce any programming or coding standards (e.g. Google C++ Style Guide)?
Not in formalised way.
Do you put effort in code maintenance. What approximately percentage of development goes into maintenance?
Yes, clearly. Difficult to answer, 10-30%.
How frequently do you reengineer or refactor code?
Often, e.g. a bit on the component and tools for each release. Our tool layer is being modernised, move from perl->python, so we are investing a big effort in thinking of better solutions there.

D. Suggestions. Please feel free to add more points and questions as well as your answers to them.

Q: Do you use 3rd party online code-analysis tools like OpenHUB?

A: Yes, see https://www.openhub.net/p/mcstas. Other softwares from SINE are also listed there, e.g. Mantid: https://www.openhub.net/p/Mantid

7.4 Reply to questionnaire: MuhRec

Answer to the questionnaire for the MuhRec software.

A. Software setup

A.1. Overview

What is the audience? What is the user base?

The main audience is the neutron imaging user community, but the nature of the algorithms makes it possible to use also with X-ray CT data.

Who contributes? Who can contribute?

Currently: Anders Kaestner (founder), Chiara Carminati. After open source release it will be possible for others to contribute.

Governance model used (e.g. does it include a management board)

There is an advisory committee set up in the frame of SINE2020. Prior to that, the development was a personal initiative.

Number of developers working on the software

2

Is the development team in a single or multiple locations?

Single

License of your software

Currently freeware, but I am about to most likely release is as LGPLv3. Would be happy to hear your comments and recommendations.

License(s) of third party libraries and software required to use the software. Are they more restrictive?

BSD-like, LGPLv3, MIT, GPL

Is user support available? What means of support?

Limited support by direct email communication.

Post-release communication with users (is there any form of stakeholder engagement?)

Currently none.

Age of the project

8 years

A.2. Requirements. How are software requirements gathered.

Who has a say in defining requirements?

The advisory board, user requests.

Do you collate requirements? How are all the requirements collated?

Before SINE2020, the project was developed based on direct (personal AK) needs. No requirements where written until now. We are working on it with the advisory board.

Do you define priorities? Deadlines?

Short term priorities are to solve problems. Longer term priorities currently only defined in the SINE2020 proposal. The advisory board are about to define priorities and deadline.

B.2. User interfaces

Do you provide a graphical user interface (GUI)? On what platform is it supported? What framework is it based on?

The tools have GUI using Qt 5.6+. Supported on

Do you provide a command line interface (CLI)?

Yes. By loading predefined parameter file that can be modified by CLI arguments.

B.4. Interoperability

Does the software interoperate with other software? How? For example, via system calls, scripting, file I/O, web services, command line?

We have tested running the tools with a python script using system calls with the CLI.

Is it possible to accommodate different data formats?

Currently, fits, tiff, and nexus are supported. Adding new formats must be done on the code level.

Are any data formats supported specifically for interoperability with some third party software?

The chosen formats are the most commonly used in neutron imaging.

Does it have an API that can be used from external software?

The modularity of the core libraries make reusing of the code possible, but is not on

specific intention.

Where do you see opportunities for and advantages of interoperability?

In a future version of the nGItool there can be a coupling to sasview

C. How is the software developed and how is software quality ensured

Development methodology used (Agile/Waterfall/other). Please provide comments. If you use agile techniques for example, which ones

Waterfall

Are version control tools used and how? What tool(s) (e.g. Git, Subversion, others)?

Currently, we use subversion but the transfer to Git is planned in 2017

Do you use any issue/ticket tracking system? What tools do you use, e.g. Trac, Jira, Redmine, or GitHub issues? How do you use it?

Github issues will be used. MuhRec already redirects to GitHub for bug reporting.

Do you use automated testing (e.g. unit, integration, unit, system, testing?

Unit testing is used, but with low coverage.

Do you use any form of acceptance testing (e.g. beta or alpha testing)?

No.

Do you have a release cycle?

About once a year.

Do you do code reviews? What tools? How often and/or how much percentage of the code?

No

Do you organize events such as code camps, hackathons, developer workshops, etc.? Please describe them.

No, The team currently sit in the same office…

Is continuous integration used and how (build servers)? What tools are used, for example Jenkins, Travis, etc.?

First build scripts are implemented with Jenkins. Will be automatized once we transfer to GitHub.

What build and configuration tools are used? For example CMake, pkg-config, make, ninja, auto-tools, Python packaging systems, etc.

QMake is used

What operating systems/platforms are supported? Are all of them fully supported?

Windows, MacOS, and Ubuntu are fully supported.

Packaging and installers. What packages do you distribute? How are they generated? Do you use any specific tool such as CPack, custom package templates, etc.?

Currently, the packaging is done by shell scripts.

Compilers supported

Windows: MSVC2015, MacOS: XCode, Linux: g++. Recommended development IDE: QtCreator

Do you use static code analysis tools (e.g. Clang tools, cpplint, Eclipse, pylint, pep8)?

No

Do you use dynamic analysis tools for debugging and/or profiling? Examples: Valgrind, Clang tools such as AddressSanitizer.

Intel Profiler long time ago.

User documentation. Do you provide for: installation, user manual, tutorials, user workshops, other learning materials (videos, are specific guides, etc.)?

Muhrec has a user manual. No learning material is provided.

What developer documentation is available?

Doxygen generated API documentation. Some few framework module guide documents.

What programming languages are used and in what proportion?

C++ (C++11) 99.9%, Python 0.1%

Are there bindings between programming languages?

None

Do you follow and/or enforce any programming or coding standards (e.g. Google C++ Style Guide)?

No, clean-up is needed

Do you put effort in code maintenance. What approximately percentage of development goes into maintenance?

10%

How frequently do you reengineer or refactor code?

Rarely.

7.5 Reply to questionnaire: SasView

Answer to the questionnaire for the SasView software.

A. Software setup

A.1. Overview
What is the audience? What is the user base?
Instrument scientists and scientists using SAS technique. The registered user community is about 50 people.
Who contributes? Who can contribute?
The contributors' team hails from 6 neutron facilities (NIST, SNS, ANISTO, ESS, PSI, ISIS). Anyone can contribute to the code by prior contacting management team.
Governance model used (e.g. does it include a management board)
The management team consists of 4 people.
Number of developers working on the software
2 full time, approx. 20 occasionally (mostly during code camps)
Is the development team in a single or multiple locations?
It is spread into 6 different institutions around the world.
License of your software
BSD
License(s) of third party libraries and software required to use the software. Are they more restrictive?
No
Is user support available? What means of support?
Yes, through the user mailing list and tutorials.
Post-release communication with users (is there any form of stakeholder engagement?)
No, free to use and redistribute
Age of the project

Started in 2006 as a part of NIH grant but in current form (as community driven project) for last 4 years

A.2. Requirements. How are software requirements gathered.

Who has a say in defining requirements?

The final say has a management team but many decisions are made during biweekly conference calls

Do you collate requirements? How are all the requirements collated?

User requirements are collated through mailing lists while other requirements are gathered on Trac issue system.

Do you define priorities? Deadlines?

The SasView road map defines long-term deadlines and priorities. Short-term deadlines are typically announced during biweekly conference calls.

B. Software architecture

B.1. Architecture

What are the general architectural principles?

The software is divided into: core module (SasCalc), model function library (sasmodels) and GUI (sasgui). There is also a minimizer module (bumps) that is separately distributed (independent of SasView).

Modularization. Are there modules, independent libraries, separate layers?

Sasmodels is a separate library (separtaly hosted on GitHub) that can be also interfaced by minimization module (bumps) directly.

Does it support plug-ins (for example for fitting functions and minimizers)?

Model functions can be supplied as plug-ins.

Is the system extensible? Is scripting supported?

Command line interface is work in progress but to some extent scripting it is already available

B.2. User interfaces

Do you provide a graphical user interface (GUI)? On what platform is it supported?

What framework is it based on?

Yes, there is a GUI. Currently it is based on wx-python but there are efforts being made to rewrite it in Qt.

Do you provide a command line interface (CLI)?

Yes, but to limited extent at the moment

B.3. Interoperability

Does the software interoperate with other software? How? For example, via system calls, scripting, file I/O, web services, command line?

The minimizer (bumps) is called from SasView by internal scripting mechanism. There is also a minimizer GUI triggered from SasView GUI.

Is it possible to accommodate different data formats?

In principle yes.

Are any data formats supported specifically for interoperability with some third party software?

Reads and writes CanSAS 1D xml and NXCansas HDF5 which provide enhanced metadata. Supports input and output in basic multi-column text files

Does it have an API that can be used from external software?

This is work in progress

Where do you see opportunities for and advantages of interoperability?

Minimizers, Model function library

C. Software development and software quality

Development methodology used (Agile/Waterfall/other). Please provide comments. If you use agile techniques for example, which ones

Code sprints and kanban implemented internally at ESS (affects 3 developers). Other developers contribute mostly during code camps.

Are version control tools used and how? What tool(s) (e.g. Git, Subversion, others)?

GIT

Do you use any issue/ticket tracking system? What tools do you use, e.g. Trac, Jira, Redmine, or GitHub issues? How do you use it?

Trac for bug reporting and future developments
Do you use automated testing (e.g. unit, integration, unit, system, testing?
Unit tests to limited extend but planed to be more in the future.
Do you use any form of acceptance testing (e.g. beta or alpha testing)?
No
Do you have a release cycle?
Each code camp should end-up with a release. So far releases has been approximately once a year.
Do you do code reviews? What tools? How often and/or how much percentage of the code?
Internally at ESS through pull requests. This is however no effective in entire SasView community.
Do you organize events such as code camps, hackathons, developer workshops, etc.? Please describe them.
Code camps – once or twice a year
Is continuous integration used and how (build servers)? What tools are used, for example Jenkins, Travis, etc.?
Build servers for Windows and OSX. Jenkins based.
What build and configuration tools are used? For example CMake, pkg-config, make, ninja, auto-tools, Python packaging systems, etc.
Python packaging: setuptools, distutils. Make for building documentation of model function library (sasmodels).
What operating systems/platforms are supported? Are all of them fully supported?
Windows 7 and OSX 10.10. The future plan includes current version and one previous.
Packaging and installers. What packages do you distribute? How are they generated? Do you use any specific tool such as CPack, custom package templates, etc.?
OSX – py2ap, Windows – pyinstaller
Compilers supported
OSX – clang, gcc. Windows – msvc, tinycc
Do you use static code analysis tools (e.g. Clang tools, cpplint, Eclipse, pylint, pep8)?
pylint, QuantifiedCode

Do you use dynamic analysis tools for debugging and/or profiling? Examples: Valgrind, Clang tools such as AddressSanitizer.
no
User documentation. Do you provide for: installation, user manual, tutorials, user workshops, other learning materials (videos, are specific guides, etc.)?
Web-based and built-in documentation and a standalone tutorial
What developer documentation is available?
Little and rather scattered
What programming languages are used and in what proportion?
Python/C++ 70%/30%
Are there bindings between programming languages?
No
Do you follow and/or enforce any programming or coding standards (e.g. Google C++ Style Guide)?
The coding standards are in preparation and are mostly collection of PEP guidelines and some SasView specific routines
Do you put effort in code maintenance. What approximately percentage of development goes into maintenance?
Entire code should be maintained
How frequently do you reengineer or refactor code?
Not too often. Primarly because of bug fixing