

Mathmet Software Quality Assurance Plan Guidance

Version: Draft v6, Date: 20/12/2022

For use with PDF Software Quality Assurance Plan Generator Draft v6

Introduction

The following document provides information that will assist with completing the Mathmet Software Quality Assurance Plan. **Sections 1 to 6 of this document correspond to sections 1 to 6 of the quality plan.** The remaining sections provide acknowledgements and references.

The aim of the plan is to **supplement**, not replace, software development procedures within Mathmet partner organisations.

It is assumed that an iterative development lifecycle will be used. However, the quality requirements listed in the plan could be met using other approaches, for example waterfall. APPENDIX I provides an example lifecycle.

Disclaimer

The Quality Assurance Tools for data, software and guidelines have been provided by the Members and Partners of the European Metrology Network for Mathematics and Statistics (Mathmet). EURAMET has no influence on its correctness and completeness and does not assume any liability for it.

Glossary

Term	Definition
Computational aim	Document providing a clear, complete and unambiguous statement of a mathematical calculation.
SWIL	Software integrity level. A value that helps quantify the risk associated with the software. A SWIL is a number between 1 and 4 , where 1 indicates the lowest level of risk and 4 the highest (typically safety-critical).
Validation	Evidence that the software can be used by the users for their specific tasks.
Verification	Evidence that the functional requirements have been met.

For further definitions, unless stated otherwise, this document refers to **BS ISO/IEC/IEEE 24765:2017 Systems and software engineering — Vocabulary** [1] for definitions of software engineering terms and the **International Vocabulary of Metrology (VIM)** [2] for definitions of terms from metrology.

Mathmet Software Quality Assurance Plan Guidance

Version: Draft v6, Date: 20/12/2022

For use with PDF Software Quality Assurance Plan Generator Draft v6

Contents

1. Software details	2
2. Document control	2
3. Software integrity level (SWIL) calculation	2
4. Software quality requirements	5
5. Other information	9
6. Version history of quality plan	9
7. Acknowledgements	9
8. References	10
APPENDIX I: Lifecycle	11
Document History	12

1. Software details

- **Software name** will be used to identify the software.
- **Brief description** can provide a brief overview of the software.
- **Developer(s)** can list the developer name(s).
- **Software location** can, for example, contain a hyperlink to a Git repository, a SharePoint folder, or a networked drive. It is important that the software is located somewhere it can be found by others than its original author(s).
- **Customer** may not be straightforward to determine. However, the matter is worth considering because it could, for example, help determine the SWIL:
 - Could the customer be a member of staff, internal to the organisation, acting as a proxy for an external organisation (for example, a funding body or an industrial client)?
 - One definition of customer is the eventual user of the software.
 - Somebody somewhere will have responsibility for funding the work for which this software is being developed.
 - There may not be a direct customer for the software itself, but someone will be the customer for the output generated by the software.
 - It is preferable that the customer is a person, rather than a generic term such as the name of an organisation.

2. Document control

An initial version of the plan should have status **DRAFT**. Before being used, the plan should be reviewed by the development team and the status changed to **ISSUED**. Also see section 6, Version history of this quality plan.

3. Software integrity level (SWIL) calculation

- A SWIL is a number between **1** and **4** where **1** indicates the lowest level of risk and **4** the highest. The SWIL shall determine the quality requirements for the software.

Mathmet Software Quality Assurance Plan Guidance

Version: Draft v6, Date: 20/12/2022

For use with PDF Software Quality Assurance Plan Generator Draft v6

- The term **SWIL** was adopted for software integrity level, rather than SIL, because of a clash of terminology with the IEC 61508 series of standards [3] where SIL is used for Safety Integrity Level. However, the IEC 61508 standards concern hardware as well as software and focus on functional safety.
- The plan refers to [1] for a definition of the term **integrity level**:
 - ...value representing project-unique characteristics, such as complexity, criticality, risk, safety level, security level, desired performance, and reliability, that define the importance of the system, software, or hardware to the user.
- Further details are provided in Table 1 below.

Table 1: Software Integrity Level (SWIL)

Software Integrity Level (SWIL)	Overview	Example
1	Not critical	Prototype / proof of concept, for example, can hardware X be used to provide measurement service Y? NOTE: Software should not be used to provide the service itself.
2	Significant	Generates results for research purposes. For example, the worst that could happen is retraction of a paper.
3	Substantial	Generates results for a measurement service, for example, numbers that will be displayed on a calibration certificate.
4	Life critical	Software that forms part of an avionics system.

- The SWIL is calculated by selecting values for the **Criticality of usage** and **Complexity of software**.
- These are also values between **1** and **4**. Tables 2 and 3 provide further details.

Table 2: Criticality of usage

CU	Criticality of usage	Explanation
1	Not critical	<ul style="list-style-type: none">• No danger of loss of income or reputation.• Short life, will not require maintenance in future
2	Significant	<ul style="list-style-type: none">• Potential for loss of income or reputation.
3	Substantial	<ul style="list-style-type: none">• Likely to lead to loss of income or reputation.
4	Life critical	<ul style="list-style-type: none">• May result in personal injury or loss of life.

Table 3: Complexity of software

CP	Complexity of software	Typical features
----	------------------------	------------------

Mathmet Software Quality Assurance Plan Guidance

Version: Draft v6, Date: 20/12/2022

For use with PDF Software Quality Assurance Plan Generator Draft v6

1	Very simple	<ul style="list-style-type: none">Elementary functionality, easy to understand.Little or no control of an external system.Simple mathematics.
2	Simple	<ul style="list-style-type: none">Simple functionality.Straightforward control of a system.Intermediate mathematics.
3	Moderate	<ul style="list-style-type: none">Large or very large programs.Difficult to modify.Complicated mathematics.
4	Complex	<ul style="list-style-type: none">Extremely complex functionality.Complex feedback systems.Very complicated mathematics.

- These values are a subjective decision. For example, simple mathematics for some could be complicated for others.
- A recommended Software Integrity Level (SWIL) is calculated from the above values, as described in table 4. The values in the cells are SWILs:

Table 4: SWIL calculation

	CS1	CS2	CS3	CS4
CU1	1	1	1	1
CU2	2	2	3	4
CU3	3	3	3	4
CU4	4	4	4	4

- Factors that justify increasing or decreasing the recommended SWIL:** The recommended SWIL can be accepted, by ticking **Is recommended SWIL suitable?** Alternatively, the SWIL could be revised. Some possible moderating factors are listed below:

Table 5: Moderating factors

Moderating factors	Possible effect on SWIL
Alternative means of verification	Decrease
Modular approach	Decrease
Suitably trained staff available	Decrease
Difficult to test	Increase
Reliant on key staff	Increase
Inexperienced staff	Increase

Mathmet Software Quality Assurance Plan Guidance

Version: Draft v6, Date: 20/12/2022

For use with PDF Software Quality Assurance Plan Generator Draft v6

Ambitious timescales	Increase
Ambitious requirements	Increase
New technology	Increase
Novel design	Increase

- Having selected **Reviewed SWIL**, tick **Confirm SWIL**. After clicking:
 - The sections for calculating the SWIL will be locked.
 - Only the software quality requirements for the confirmed SWIL will be displayed.
 - An asterisk will be displayed next to the title of the requirements that are mandatory.
- If the software is determined to be SWIL 4 **then seek guidance from safety critical software experts.**

4. Software quality requirements

This section of the plan lists the quality requirements for the selected SWIL. Mandatory requirements are indicated with an asterisk.

The plan supports rich and long text and could be sufficient for storing details of the requirements for **very** small pieces of software. However, it is usually better practice to provide a short explanation and a link to a working document, stored in a permanent place that can be made accessible to others than the original developer(s).

The quality requirements for each SWIL are listed as a series of tables. These tables use the following key:

Table 6: Key for following tables

X	Not required
R	Recommended
M	Mandatory

The tables are listed below, followed by some guidance on meeting some of the quality requirements.

Table 7: User requirements

Quality Requirement	SWIL 1	SWIL 2	SWIL 3	SWIL 4
Documented user requirements	M	M	M	M
Review by team	R	M	M	M
Review by suitably qualified independent person	X	X	R	M
Review by customer or proxy	M	M	M	M

Table 8: Functional requirements

Mathmet Software Quality Assurance Plan Guidance

Version: Draft v6, Date: 20/12/2022

For use with PDF Software Quality Assurance Plan Generator Draft v6

Quality Requirement	SWIL 1	SWIL 2	SWIL 3	SWIL 4
Documented functional requirements	R	M	M	M
Traceable requirements (from user requirements through functional requirements to code and test plan)	X	M	M	M
Review by team	R	M	M	M
Review by suitably qualified independent person	X	X	R	M

Table 9: Design

Quality Requirement	SWIL 1	SWIL 2	SWIL 3	SWIL 4
Informal design	R	M	M	M
Clear and well-structured documented design	X	R	M	M
Review by team	R	M	M	M
Review by suitably qualified independent person	X	X	R	M

Table 10: Coding

Quality Requirement	SWIL 1	SWIL 2	SWIL 3	SWIL 4
Header to identify program name, author, date and version number	M	M	M	M
Program history	R	M	M	M
Coding guidelines	X	M	M	M
Review by team	R	R	M	M
Review by suitably qualified independent person	X	X	R	M

Table 11: Verification

Quality Requirement	SWIL 1	SWIL 2	SWIL 3	SWIL 4
Module testing as coding progresses	R	M	M	M
Verification of complete software against functional requirements	R	M	M	M
Review by team	R	R	M	M
Review by suitably qualified independent person	X	X	R	M

Table 12: Validation

Quality Requirement	SWIL 1	SWIL 2	SWIL 3	SWIL 4
Validation against user requirements	R	R	M	M
Review by team	M	M	M	M
Review by suitably qualified independent person	X	X	R	M

Mathmet Software Quality Assurance Plan Guidance

Version: Draft v6, Date: 20/12/2022

For use with PDF Software Quality Assurance Plan Generator Draft v6

Table 13: Delivery, use and maintenance

Quality Requirement	SWIL 1	SWIL 2	SWIL 3	SWIL 4
Version control on release	R	M	M	M
Version control before release	X	R	M	M
Bug tracking/error logging	X	R	M	M
Traceability of output	R	M	M	M
User documentation	R	M	M	M

The following points list some matters to consider when developing software using this plan:

- **User requirements: Documented user requirements**

User requirements are a statement of **what** is required from the software, not **how** it is going to meet these requirements.

Enter a link to the document that contains the user requirements, or to a folder where the documents are held. For smaller pieces of software, it may be possible to enter the requirements directory into the plan.

Matters to consider in user requirements, **before development begins**, include:

- User friendly GUI
- Data/statistical analysis
- Data visualisation and graphics
- Notifications and alerts
- Clear documentation for users

The Mathmet quality assurance tools provide a template that can help with requirements capture. If an alternative template is considered more appropriate, use that instead.

- **User requirements: Review: team / independent / customer**

Enter link(s) to evidence of review.

- **Functional requirements: Documented functional requirements**

Non-trivial **mathematics** is highly likely to underpin most of the software developed with the assistance of this plan. A clear, complete and unambiguous statement of the mathematics will allow it to be verified without having to examine any code. It must **not** be necessary to examine the code of even the shortest script to determine the mathematics implemented.

An online database of documents called **computational aims** [4], developed as part of the EURAMET Traceability for computationally intensive metrology (TraCIM) project [5], provides an example of specifying mathematical calculations.

Other matters to consider in functional requirements, **before development begins**, include report generation and input/output data formats. For software that will form part of a **calibration system** also consider:

- Calculations for the calibration of a measuring instrument,
- Determination of uncertainties
- Easy upload of measurement and calibration data

Mathmet Software Quality Assurance Plan Guidance

Version: Draft v6, Date: 20/12/2022

For use with PDF Software Quality Assurance Plan Generator Draft v6

- Recording of calculation results
- Analysis of trends on historical data (measuring instruments, standards and calibrations items)
- Notifications of acceptance criteria and measurement requirements
- Publication of calibration certificates

- **Functional requirements: Traceable requirements**

Functional requirements should be labelled in a way that allows them to be traceable from the user requirements to the code and tests that help verify the code.

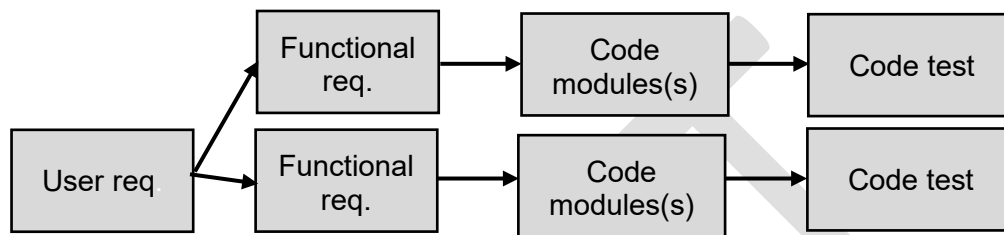


Figure 1: Requirements traceability

- **Design: Informal design / Clear and well-structured documented design**

Enter link(s) to document(s) providing the software design. For much of the software developed with the assistance of this plan, a simple block diagram may well be sufficient for both informal and well-structured documented design. For example, software that consists of a few MATLAB scripts could have its design documented with a block diagram illustrating which script calls which script **and which is the main script**.

Consider the provenance of packages and libraries. For example, a richly featured, but new and experimental, library may be appropriate for SWIL 1 or 2 software but not SWIL 3 or 4. There are often no “right” or “wrong” answers, just decisions to **made, documented and reviewed**.

Other matters to consider include:

- Modular structure
- Ease of maintenance
- Ability to incorporate new functionality

- **Coding: Header to identify program name, author, date and version number**

Variables that identify the software name, version and date of release are important for even the smallest script. It is important the user is left in no doubt as to the name and version number of the software being run. Such information is key to traceability of output, as will be noted in **delivery, use and maintenance** below.

- **Coding: Coding guidelines**

The use of **guidelines**, such as the PEP 8 – Style Guide for Python Code [6] or other guidelines [7] defined or recommended within an organisation will aid development of cleaner, neater, more easily readable, and therefore maintainable, code.

Enter link(s) to guidelines(s) used.

- **Verification of complete software against functional requirements**

Mathmet Software Quality Assurance Plan Guidance

Version: Draft v6, Date: 20/12/2022

For use with PDF Software Quality Assurance Plan Generator Draft v6

Enter a link to evidence that the functional requirements have been met. For software developed with the assistance of this plan, **evidence that non-trivial mathematics has been implemented correctly** is a key component of verification.

As noted in **functional requirements** a clear, concise and unambiguous statement of the mathematics eases verification. An alternative implementation of selected calculations could be implemented using an alternative platform such as a spreadsheet.

- **Validation against user requirements**

Enter a link to evidence that the user requirements have been met. One definition from [1] states “Validation demonstrates that the system can be used by the users for their specific tasks”.

- **Delivery, use and maintenance: Version control**

Enter details of how version control will be achieved, for example using a tool such as Git or Subversion. For very simple pieces of software an appropriate file or folder naming convention may be sufficient.

- **Delivery, use and maintenance: Traceability of output**

Enter details of how the outputs generated by the software, for example results to be presented in customer certificates or research papers, can be traced to the **name and version** of the software that generated them. Other information, such as date and time of execution, identifier of operator/user and location of raw data may also be necessary.

Reproducibility of results [8] can be made easier by making such data available.

5. Other information

- This section provides details such as the platform, for example, operating system and hardware, on which the software will run. Any specific requirements for the software, for example minimum hardware requirements, could also be listed in this section.
- One possible difficulty could be presented by **Responsibilities for testing**. As discussed in section 1, who the customer is may not necessarily be easy to determine.
- Selecting **Mathematical Area(s)** and **Metrology Area(s)** will be helpful when considering how the software should be verified and validated. Such information is also useful documentation.

6. Version history of quality plan

- This section should be straightforward to complete, although consideration is required as to who will approve ISSUED versions of the plan.
- If sufficient space has not been provided in this section, a supplementary document can be created.

7. Acknowledgements

- The European Metrology Network for Mathematics and Statistics is supported by the Joint Network Project ‘Support for a European Metrology Network for mathematics and statistics’ (18NET05 MATHMET). The project 18NET05 MATHMET has received funding from the EMPIR programme co-financed by the Participating States and from the European Union’s Horizon 2020 research and innovation programme.

Mathmet Software Quality Assurance Plan Guidance

Version: Draft v6, Date: 20/12/2022

For use with PDF Software Quality Assurance Plan Generator Draft v6

- Many thanks to the Quality Assurance team of the National Physical Laboratory, for their advice and assistance.

8. References

[1] BS ISO/IEC/IEEE 24765:2017 Systems and software engineering — Vocabulary. Online version retrieved 14/12/2022 from IEEE:

<https://www.computer.org/sevocab>

[2] BIPM, IEC, IFCC, ILAC, ISO, IUPAC, IUPAP, and OIML. International vocabulary of metrology | Basic and general concepts and associated terms (VIM). Joint Committee for Guides in Metrology, JCGM 200:2012. (3rd edition). URL (Retrieved 14/12/2022 from BIPM): https://www.bipm.org/documents/20126/2071204/JCGM_200_2012.pdf/f0e1ad45-d337-bbeb-53a6-15fe649d0ff1

[3] What is IEC 61508? Retrieved 14/12/2022 from the 61508 Association:

<https://www.61508.org/knowledge/what-is-iec-61508.php>

[4] TraCIM computational aims database. Retrieved 14/12/2022 from NPL:

<http://www.tracim-cadb.npl.co.uk/>

[5] TraCIM project homepage. Retrieved 14/12/2022 from PTB:

<https://www.ptb.de/emrp/tcim.html>

[6] PEP 8 – Style Guide for Python Code. Retrieved 14/12/2022 from python.org:

<https://peps.python.org/pep-0008/>

[7] Making the Best Use of C. Retrieved 14/12/2022 from gnu.org:

https://www.gnu.org/prep/standards/html_node/Writing-C.html

[8] 1,500 scientists lift the lid on reproducibility. Retrieved 14/12/2022 from Nature:

<https://www.nature.com/articles/533452a>

For use with PDF Software Quality Assurance Plan Generator Draft v6

```

graph TD
    Start((Start)) --> J1{+}
    J1 --> A1[Determine mathematical area, metrological area]
    J1 --> A2[Assess complexity of calculation  
(e.g. discrete, partial differential)]
    A1 --> J2{+}
    A2 --> J2
    J2 --> A3[Risk Analysis]
    A3 --> A4[Develop Software Quality Management (SQM) plan]
    A4 --> J3{ }
    J3 --> A5[Review SQM plan]
    A5 --> J3
    J3 --> J4{+}
    J4 --> A6[Develop verification plan]
    J4 --> A7[Requirements capture, develop computational aim]
    A6 --> J5{ }
    A7 --> J6{ }
    J5 --> A8[Review verification plan]
    A8 --> J5
    J6 --> A9[Review requirements]
    A9 --> J6
    J5 --> J7{+}
    J6 --> J7
    J7 --> A10[Design and code]
    A10 --> J8{ }
    J8 --> A11[Verify software]
    A11 --> J8
    J8 --> J9{ }
    J9 --> A12[Validate software / final review]
    A12 --> J9
    J9 --> A13[Release]
    A13 --> A14[Maintain]
    A14 --> End((End))
    A14 --> J3
    A14 --> J6
    A14 --> J9
  
```

The flowchart illustrates the Software Quality Management (SQM) process, organized into three main phases: Plan, Do, and Check + Act.

- Plan Phase:**
 - Starts with a green circle labeled "Start".
 - Flow splits into two parallel tasks: "Determine mathematical area, metrological area" and "Assess complexity of calculation (e.g. discrete, partial differential)".
 - These tasks merge at a junction (+).
 - Flow proceeds to "Risk Analysis".
 - Next is "Develop Software Quality Management (SQM) plan".
 - A decision diamond follows, with a feedback loop to "Review SQM plan" and a path to the next junction (+).
 - The next junction (+) splits into two parallel tasks: "Develop verification plan" and "Requirements capture, develop computational aim".
 - Each task has a feedback loop: "Develop verification plan" leads to "Review verification plan", and "Requirements capture, develop computational aim" leads to "Review requirements".
 - Both review tasks merge at a junction (+).
- Do Phase:**
 - The flow proceeds to "Design and code".
 - A decision diamond follows, with a feedback loop to "Verify software" and a path to the next junction.
- Check + Act Phase:**
 - The flow proceeds to "Validate software / final review".
 - A decision diamond follows, with a feedback loop to "Validate software / final review" and a path to "Release".
 - The flow proceeds to "Release", then "Maintain", and finally "End" (red circle).
 - There are three feedback loops from the "Maintain" stage back to the Plan phase:
 - From "Maintain" back to the junction before "Develop Software Quality Management (SQM) plan".
 - From "Maintain" back to the junction before "Develop verification plan" and "Requirements capture, develop computational aim".
 - From "Maintain" back to the junction before "Design and code".

Figure 1: Example software lifecycle

Mathmet Software Quality Assurance Plan Guidance

Version: Draft v6, Date: 20/12/2022

For use with PDF Software Quality Assurance Plan Generator Draft v6

Document History

Version	Date	Revised by	Change(s)	Approved by
Draft v1	10/05/2022	K.Lines, NPL	INITIAL RELEASE	N/A
Draft v2	22/08/2022	K.Lines, NPL	Correct typos. Add "Review by suitably qualified independent person" to tables 10 and 11.	N/A
Draft v3	29/08/2022	K.Lines, NPL	Added feedback from WP2 partners. Added further details about quality requirements. Release for further review by WP2 partners and other interested parties.	N/A
Draft v4	16/11/2022	K.Lines, NPL	Added feedback from EURAMET quality managers. Replaced references to quality management system with references to quality assurance tools. Removed logos and added disclaimer.	N/A
Draft v5	14/12/2022	K.Lines, NPL	Added feedback from Mathmet Chair. Modified title and added further details to acknowledgements.	N/A
Draft v6	20/12/2022	K.Lines, NPL	Added reference to latest version of Software Quality Assurance Plan Generator.	N/A