

Mathmet Software Quality Assurance Plan Guidance

For use with PDF Data Quality Assurance Plan Generator, Version 1.0.0

Disclaimer

No representation is made, nor warranty given that this document or the information contained in it will be suitable for any particular purpose. In no event shall EURAMET, the authors or anyone else involved in the creation of the document be liable for any damages whatsoever arising out of the use of the information contained herein. The parties using the guide shall indemnify EURAMET accordingly.

EURAMET e.V., Bundesallee 100, 38116 Braunschweig, Germany

Phone: +49 531 592 1960 Fax: +49 531 592 1969 E-mail: secretariat@euramet.org

www.euramet.org



MATHMET

Authorship and Imprint

This document was developed by the EURAMET e.V., European Metrology Network for Mathematics and Statistics

Authors: Jean-Laurent Hippolyte (NPL, UK), Keith Lines (NPL, UK), Indhu George (NPL, UK)

Version

1.0.0 (07/2025)

Official language

The English language version of this document is the definitive version.

Copyright

The copyright of this publication (Mathmet Software Quality Assurance Plan Guidance, Version 1.0.0 – English version) is held by © EURAMET e.V. 2025. The text may not be copied for resale and may not be reproduced other than in full.

Acknowledgements

This document was reviewed and revised by the EURAMET e.V., Technical Committee for Interdisciplinary Metrology.

Reviewers: Giacomo Lanza, (PTB, Germany), João Gregorio (NPL, UK)

The European Metrology Network for Mathematics and Statistics was 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.

The Mathmet quality assurance tools were developed further by the National Physical Laboratory's Data Science department as part of Data Science's software and data quality assurance for metrology (NMS) project 2024 – 2025.

This work was developed from the quality management system of the National Physical Laboratory, UK. See references [1] and [2] for further details.

CONTENTS

GLOSSARY	5
INTRODUCTION	7
1. SOFTWARE OVERVIEW	7
Q1.1 Software name	7
Q1.2 Brief description	7
Q1.3 Developer(s)	7
Q1.4 Customer	7
Q1.5 Location of software and documentation	8
2. VERSION CONTROL OF PLAN	8
Q2.1 Status of plan	8
Q2.2 Version of plan	8
Q2.3 Date of version	8
Q2.4 Plan author(s)	8
3. SOFTWARE INTEGRITY LEVEL	9
Q3.1 Criticality of usage	10
Q3.2 Complexity of software	10
Q3.3a Is recommended SWIL suitable?	11
Q3.3b Factors that justify increasing or decreasing the recommended SWIL	11
4. USER REQUIREMENTS	12
Q4.1 Documentation of user requirements	12
Q4.2 Review by team	12
Q4.3 Review by suitably qualified independent person	12
Q4.4 Review by customer or proxy	12
5. FUNCTIONAL REQUIREMENTS	13
Q5.1 Documentation of functional requirements	13
Q5.2 Traceability of requirements	13
Q5.3 Review by team	14
Q5.4 Review by suitably qualified independent person	14
6. DESIGN	14
Q6.1 Informal design	14
Q6.2 Clear and well-structured Documentation of design	14
Q6.3 Review by team	14
Q6.4 Review by suitably qualified independent person	14
7. CODING	15
Q7.1 Identify software name, author, date and version number	15
Q7.2 Program history	15
Q7.3 Coding guidelines	15
Q7.4 Review by team	15
Q7.5 Review by suitably qualified independent person	15
8. VERIFICATION	16

Q8.1 Module testing as coding progresses	16
Q8.2 Verification of complete software against functional requirements	16
Q8.3 Review by team	16
Q8.4 Review by suitably qualified independent person	16
9. VALIDATION	16
Q9.1 Validation of complete software against user requirements.....	16
Q9.2 Review by team	16
Q9.3 Review by suitably qualified independent person	16
10. DELIVERY, USE AND MAINTENANCE.....	17
Q10.1 Version control on release	17
Q10.2 Version control before release	17
Q10.3 Enhancement requests / bug tracking	17
Q10.4 Traceability of output.....	17
Q10.5 User documentation.....	17
11. PLATFORM ETC.	18
Q11.1 Platform	18
Q11.2 Type.....	18
Q11.3 Languages(s).....	18
Q11.4 Responsibilities for testing	18
Q11.5 Backup regime	18
Q11.6 Release rules	18
12. CONFIGURATION AND MAINTENANCE.....	18
Q12.1 Configuration management.....	18
Q12.2 Maintenance plan.....	18
13. OTHER INFORMATION: MATHEMATICAL AREA(S) AND METROLOGY AREA(S)	19
Q13.1 Mathematical area(s)	19
Q13.2 Metrology area(s).....	19
14. VERSION HISTORY OF QUALITY PLAN.....	19
APPENDIX II: LIFECYCLE	23
DOCUMENT HISTORY	24

GLOSSARY

Term	Definition	Source
Customer	Person or organisation that could or does receive a product or a service that is intended for or required by this person or organisation. A customer can be internal or external to the organisation.	[3]
Computational aim	Document providing a clear, complete and unambiguous statement of a mathematical calculation.	[4]
Development team	Team that develops or maintains software.	[5]
Quality	The degree to which a set of inherent characteristics of an object fulfils requirements.	[3]
Release	Put into use in providing a service, delivered to a customer or used to generate output for inclusion in work to be delivered to a customer (e.g. project deliverables, papers, reports etc.).	N/A
Requirement	Need or expectation that is stated, generally implied or obligatory.	[3]
Requirements traceability / Traceability of requirements	Discernible association between a requirement and related requirements, implementations, and verifications.	[5]
Risk analysis	Systematic use of available information to identify hazards and to estimate the risk.	[6]
Software	Shall refer to any software type.	N/A
Software type	Includes spreadsheets, databases, macros, scripts, programs and web applications.	N/A
Software integrity level (SWIL)	A risk assessment metric that accounts for complexity and criticality levels. 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).	Section 3. Software Integrity Level
Trustworthy	Appropriately addresses safety, reliability, availability, resilience and security issues. NOTE: Reference [3] refers to these issues as the five facets of trustworthiness.	[7]

Term	Definition	Source
Uncertainty budget	Statement of a measurement uncertainty, of the components of that measurement uncertainty, and of their calculation and combination.	[8]
Validation	<p>Confirmation, through the provision of objective evidence, that the requirements for a specific intended use or application have been fulfilled.</p> <p>NOTE: For the Mathmet Software Quality Assurance Plan, validation focuses on user requirements (see sections 5 and 9).</p> <p>NOTE: In some instances, the subjective opinion of a knowledgeable user could be the most pragmatic means of obtaining evidence. See section 9 for further details.</p>	[3]
Verification	<p>Confirmation, through the provision of objective evidence, that specified requirements have been fulfilled.</p> <p>NOTE: For the Mathmet Software Quality Assurance Plan, verification focuses on functional requirements (see sections 4 and 8).</p>	[3]

For further definitions, unless stated otherwise, this document refers to **BS ISO/IEC/IEEE 24765:2017 Systems and software engineering — Vocabulary** [5] for definitions of software engineering terms and the **International Vocabulary of Metrology (VIM)** [8] for definitions of terms from metrology.

INTRODUCTION

The following document provides information to assist with completing a Mathmet Software Quality Assurance Plan. The associated interactive PDF tool supports documenting the people, procedures and activities performed to fulfil predefined quality objectives during the development and maintenance of software within the metrology community and wider scientific domains, in which quality and trustworthiness are essential.

Sections 1 to 12 of this document correspond to sections 1 to 12 of the PDF tool. The remaining sections contain references and appendices providing further information.

The aim of the plan is to supplement, not replace, software development procedures within Mathmet partner organisations. The completed plan provides guidance for developers and can help guide software quality audits.

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 or agile. APPENDIX II provides an example lifecycle.

Training materials for the Mathmet Software Quality Assurance Plan and other Quality Assurance Tools (QAT) are available on the Mathmet website [9]. Further details of the approach taken in this plan are provided in reference [10].

1. SOFTWARE OVERVIEW

This section provides general information about the software development project.

Q1.1 Software name

Enter an unambiguous name for the software. The naming convention used will be determined by the developer(s)' organisation(s) and the project(s) for which the software is being developed.

Q1.2 Brief description

Describe, as briefly as possible, the intended tasks the software will perform, users and equipment (for example, hardware). Also include any other relevant information (for example, programming languages or operating systems). See definition of **user context** in reference [5].

These subjects should be covered in more detail in the documentation to be written during software development. But an initial description, comparable to an abstract for a document, will help determine the Software Integrity Level (SWIL) in section 3.

Q1.3 Developer(s)

Enter the name(s) and, if required, role(s) and contact details of the development team. Team members may have different roles: for example, technical lead, programmer(s), tester(s). For smaller teams (a team might have a single member) members could have multiple roles.

Q1.4 Customer

Enter the name of the individual or organisation who accepts delivery of the software or output generated by the software. Examples of such output include calibration certificates or results presented in an academic paper. It may not be easy to determine who the customer is. However,

the matter is worth considering because it could, for example, help determine the SWIL. See definition of **customer** in the glossary at the start of this document.

Usually, it is preferable to list the customer as a person rather than a generic term such as an organisation name. More specifically, this person might be:

- A staff member in an organisation developing the software, acting as a proxy for an external organisation (for example, a funding body or an industrial client).
- Someone with responsibility for funding a project 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.

Q1.5 Location of software and documentation

Provide the location where the software, including source code, and documentation is stored. For example, provide a hyperlink to a Git repository, a shared folder or a networked drive.

It is **vital** that the source code and documentation is **not** stored somewhere it can only be found easily by the original developer(s). Finding the correct versions of the source code and documentation could be a significant challenge after the developers have left the organisation. Other matters to consider regarding location are listed in section 2.2 of reference [10].

2. VERSION CONTROL OF PLAN

This section of the software quality assurance plan concerns the plan itself, not the software and not the interactive PDF tool.

Q2.1 Status of plan

Select the status. An initial version of the plan should have status DRAFT. Before being followed, the plan should be reviewed by the development team and the status changed to ISSUED.

Q2.2 Version of plan

Enter the version number of **the plan**. The version numbering convention used will be determined by the developer(s)' organisation(s) and the project(s) for which the software is being developed.

Q2.3 Date of version

Select the date that this version of the plan was completed.

Q2.4 Plan author(s)

Enter the name(s) of the plan's author(s). It is recommended to provide contact details, such an email address, for at least one of the authors.

3. SOFTWARE INTEGRITY LEVEL

A risk analysis results in the calculation of a Software Integrity Level (SWIL). The SWIL is a numeric value, between 1 and 4, that determines the quality requirements for the software.¹

The term **integrity level** conforms to reference [5]:

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.

Examples are provided in Table 1.

Table 1: Software Integrity Level (SWIL) and examples.

Software Integrity Level (SWIL)	Overview	Example
1	Not critical	<ul style="list-style-type: none"> Prototype / proof of concept. For example could new hardware, controlled by this software, help provide a measurement service? NOTE: SWIL1 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 using two other parameters, **criticality of usage** and **complexity of software**, which also take values between 1 and 4 (where 1 indicates the lowest level of risk and 4 the highest). The selection of values for these parameters is to some extent subjective. However, factors such as the following could help determine these values and should be noted:

- Is the software being developed either by, or in close collaboration with, expert mathematicians or statisticians or scientists?
- Is it anticipated that the software would consist of a large number of modules, or call a large number of libraries?
- Is the software intended to be used by technically knowledgeable operators?

¹ The acronym **SWIL**, rather than SIL, was adopted because of a clash of terminology with the IEC 61508 series of standards [11], where SIL is used for Safety Integrity Level, a term which focuses on functional safety and concerns hardware as well as software.

Q3.1 Criticality of usage

Evaluate the criticality of usage (CU) of the software, i.e. the consequences of customers using software of inappropriate quality particularly on business/reputation.

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	Potential for loss of income or reputation.
3	Substantial	Likely to lead to loss of income or reputation.
4	Life critical	May result in personal injury, loss of life or damage to the environment.

Q3.2 Complexity of software

Evaluate the anticipated complexity of the software. Software complexity (CP) can be defined considering one or more of: (a) the complexity of the mathematics implemented (b) the anticipated size of the source code, perhaps as lines of code or function points [5] (c) whether control of an external system is involved.

Table 3: Complexity of software

CP	Complexity of software	Explanation, typical features
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	Life critical	<ul style="list-style-type: none"> Extremely complex functionality. Complex feedback systems. Very complicated mathematics.

Q3.3a Is recommended SWIL suitable?

After risk has been analysed, the interactive PDF tool will compute a recommended SWIL value according to the rules listed in table 4. The values in the cells are SWILs:

Table 4: Calculating the Software Integrity Level.

	CP 1	CP 2	CP 3	CP 4
CU 1	1	1	1	1
CU 2	2	2	3	4
CU 3	3	3	3	4
CU 4	4	4	4	4

This recommended SWIL can be accepted or could be revised by the development team. For example, if inexperienced software developers are involved the team can decide to increase the SWIL. If the SWIL value is revised, then the factors that justify increasing or decreasing the recommended SWIL value must be documented.

Q3.3b 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 in table 5.

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
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. USER REQUIREMENTS

This section and the following sections up to and including section 13, list the tasks and documentation required for the application scenario quantified by the SWIL.

The plan supports long and formatted text which could be sufficient for storing details of the requirements for small pieces of software, such as those that consist of a small number of scripts. 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).

Q4.1 Documentation of 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.

Q4.2 Review by team

Enter link(s) to evidence of review or a description of how evidence of review will be captured and acted on. Examples of evidence are a commented document, an email trail (copied to a project folder, not stored within the email system itself) or comments within a Git repository.

Q4.3 Review by suitably qualified independent person

Enter link(s) to evidence of review or a description of how evidence of review will be captured and acted on. The reviewer shall be independent of the team that developed the software but could be a colleague within the organisation where the software was developed.

Q4.4 Review by customer or proxy

Enter link(s) to evidence of review or a description of how evidence of review will be captured and acted on.

5. FUNCTIONAL REQUIREMENTS

In this plan, function does not mean a software module, for example, a Python script, a C function, or a database form. It means a task, action, or activity that must be accomplished to achieve a desired outcome.

Q5.1 Documentation of functional requirements

Non-trivial **mathematics** is highly likely to underpin most of the software developed with the assistance of this plan generator. 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**, developed as part of the EURAMET “Traceability for computationally intensive metrology” (TraCIM) project [4], 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 uncertainty budgets [8].
- Easy upload of measurement and calibration data.
- Recording of calculation results.
- Analysis of trends on historical data (measuring instruments, standards and calibration items).
- Notifications of acceptance criteria and measurement requirements.
- Publication of calibration certificates.

Q5.2 Traceability of 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. Traceable requirements can help ensure mathematics is implemented correctly. Describe how functional requirements will be made traceable.

See definition of **requirements traceability** in the glossary at the start of this document.

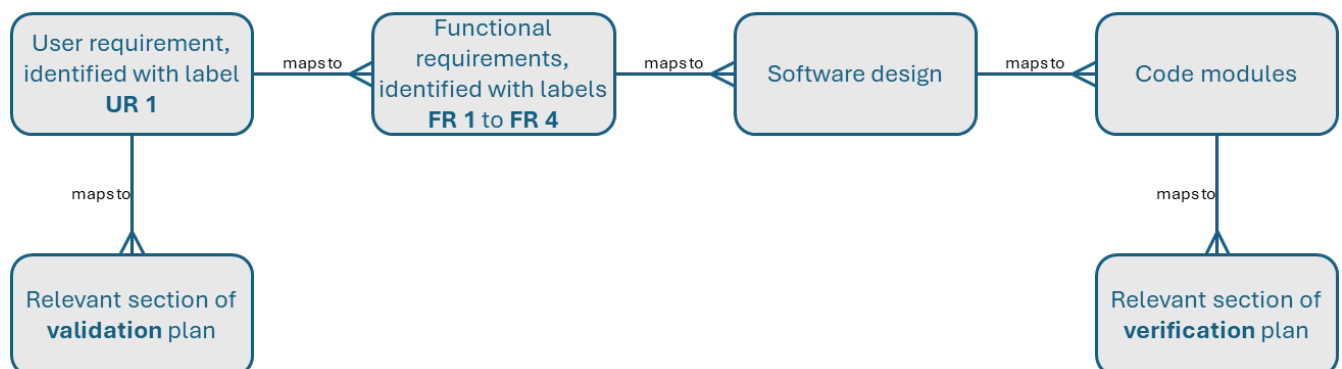


Figure 1: Example of traceability of requirements

Q5.3 Review by team

Enter link(s) to evidence of review or a description of how evidence of review will be captured and acted on.

Q5.4 Review by suitably qualified independent person

Enter link(s) to evidence of review or a description of how evidence of review will be captured and acted on. The reviewer shall be independent of the team that developed the software but could be a colleague within the organisation where the software was developed.

6. DESIGN

For much of the software developed with the assistance of this plan, a simple block diagram may be sufficient for both informal and well-structured Documentation of design (see below). For example, software that consists of a few scripts could have its design Documentation of using a block diagram illustrating the order in which scripts are called **and which is the main script**.

For other software, particularly SWIL 3 or 4 software, formal design languages such as the Unified Modelling Language (UML) [12] or the Systems Modelling Language (SysML) [13] could be helpful.

Also 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 be taken, Documentation of and reviewed.

Other matters to consider include:

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

Q6.1 Informal design

Enter link(s) to document(s) providing the software design.

Q6.2 Clear and well-structured Documentation of design

Enter link(s) to document(s) providing the software design.

Q6.3 Review by team

Enter link(s) to evidence of review or a description of how evidence of review will be captured and acted on.

Q6.4 Review by suitably qualified independent person

Enter link(s) to evidence of review or a description of how evidence of review will be captured and acted on.

7. CODING

This section of the plan outlines matters to be considered while coding.

Q7.1 Identify software name, author, date and version number

Identifying the software name, version and date of release is important for even the smallest piece of software. The user must be sure which version of the software being run. This information is also key to the traceability of output, as will be noted in **delivery, use and maintenance** below.

In this section of the plan, indicate how this identification will be achieved. For example:

- Include display the software name and version number on start-up, even for scripts consisting of a few lines. The version numbering convention used will be determined by the developer(s)' organisation(s) and the project(s) for which the software is being developed. For example, semantic versioning [14] (<major>.<minor>.<patch>) could be used.
- If the software has a Graphical User Interface, display the software name and version number prominently.
- A header comment within the source code can be used to identify the required information,

Q7.2 Program history

Enter details of how modifications to be the code will be recorded. A header comment in the code is one of achieving this aim.

Q7.3 Coding guidelines

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

Guidelines, such as the PEP 8 – Style Guide for Python Code [15], GNU Coding Standards [16] or guidelines internal to an organisation will aid development of neater, more easily readable, and maintainable, code. Guidelines could be mandated or recommended within an organisation.

Q7.4 Review by team

Enter link(s) to evidence of review.

Q7.5 Review by suitably qualified independent person

Enter link(s) to evidence of review.

8. VERIFICATION

This section helps ensure that the software is working correctly. See definition of **verification** in the glossary at the start of this document.

Q8.1 Module testing as coding progresses

Informal notes in a project log could be sufficient for smaller pieces of software. For larger pieces of software, a formal test plan could be required. The development team must make the appropriate decision.

Q8.2 Verification of complete software against functional requirements

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, for example, using a spreadsheet.

Q8.3 Review by team

Enter link(s) to evidence of review.

Q8.4 Review by suitably qualified independent person

Enter link(s) to evidence of review.

9. VALIDATION

This section helps ensure user requirements have been met. See definition of **validation** in the glossary at the start of this document.

Q9.1 Validation of complete software against user requirements

Enter a link to evidence that user requirements have been met. Traceable requirements, see **Q5.2**, assist with providing this evidence.

In some instances, such evidence may be the subjective opinion of the customer. For example, if the customer is a metrologist with years of experience providing a measurement service their opinions will be vital as to whether a user requirement has been met.

Q9.2 Review by team

Enter link(s) to evidence of review.

Q9.3 Review by suitably qualified independent person

Enter link(s) to evidence of review.

10. DELIVERY, USE AND MAINTENANCE

This section covers matters that help the software be useable in the longer term. Even smaller pieces of software could benefit, as it can aid reproducibility and reusability.

Q10.1 Version control on release

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

Q10.2 Version control before release

As above, expect maintain version control before software is released.

Q10.3 Enhancement requests / bug tracking

Enter details of how enhancement requests and bug reports will be logged and tracked. For example, Git provides features for issue tracker. In some instances, as considered appropriate by the development team, email trails (copied to a project folder, not stored within the email system itself) may be appropriate for this purpose.

Q10.4 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 [17] can be made easier by making such data available.

Q10.5 User documentation

Explain how user documentation will be provided. For example, a document could be written covering matters such as the following:

- Program identifier/name and version number
- Install/uninstall guide
- Overview of physics / mathematics etc.
- Instructions for operation
- How to report enhancement requests / bugs

For very small pieces of software, such as those that consist of a small number of scripts, a README file could be sufficient.

11. PLATFORM ETC.

This section contains information concerning the practical development of the software. It documents technical details on programming and release management, strategical and security-related decisions and responsibilities.

Q11.1 Platform

Enter details of hardware device and/or associated operating system, or virtual environment, on which software can be installed or run. The software could also run on a cloud platform. See definitions of **platform** and **platform as a service (PaaS)** in reference [5] for further details.

Q11.2 Type

Select appropriate value. See definition of **software type** in the glossary at the start of this document for further details.

Q11.3 Languages(s)

Enter names of programming language(s) that will be used. For example, C, C#, Fortran, LabVIEW, MATLAB, Python etc.

Q11.4 Responsibilities for testing

As noted in **Q1.4**, it may not necessarily be easy to determine who is the customer.

Q11.5 Backup regime

Enter details of how files will be backed up. Note that this matter is not same as version control. Reference [5] describes backup as “system, component, file, procedure, or person available to replace or help restore a primary item in the event of a failure or externally caused disaster”.

Q11.6 Release rules

See definition of **release** in the glossary at the start of this document. A list of instructions on how to release the software, such as a checklist, can help avoid errors in releasing the software.

12. CONFIGURATION AND MAINTENANCE

Q12.1 Configuration management

Enter details of how components of the overall system will be managed, both hardware and software. E.g., if external libraries are required how will user know which versions?

Q12.2 Maintenance plan

If necessary, provide further outline of how issues described in section 10 will be implemented.

13. OTHER INFORMATION: MATHEMATICAL AREA(S) AND METROLOGY AREA(S)

Selecting Mathematical Area(s) and Metrology Area(s) will be helpful when considering how the software should be verified and validated.

Q13.1 Mathematical area(s)

The list of mathematical areas was determined by the EURAMET “Traceability for computationally intensive metrology” (TraCIM) project [4].

Q13.2 Metrology area(s)

The list of metrology areas is derived from the list of EURAMET Technical Committees [18].

14. VERSION HISTORY OF QUALITY PLAN

Updating the plan at appropriate stages during the software development project is recommended. For example, an initial DRAFT version can be developed, reviewed by the project team (in particular, confirming that SWIL is appropriate) and an ISSUED version released and used. New versions could also be released after documentation has been developed or reviews completed.

VERSION

Enter the version number of the plan.

DATE

Select the date that this version of the plan was completed.

AUTHOR(S)

Enter the names of all the plan’s/software’s authors and at list the contact details (e-mail) of a selected contact person.

CHANGE(S) FROM PREVIOUS VERSION

List the features that have been updated from the previous version. Enter details of the change(s) in this version of the plan. For the first version of this plan enter **initial version**.

APPROVER(S)

Enter the names of all leaders or stakeholders who are designated to approve the project progress and the plan update.

REFERENCES

- [1] *Wichmann, B., Parkin, G., Barker, R., “Software Support for Metrology Best Practice Guide No. 1. Validation of Software in Measurement Systems”, NPL Report DEM-ES 014.*
[Online]. Available: http://resource.npl.co.uk/docs/science_technology/scientific_computing/ssfm/documents/ssfmbpg1.pdf
- [2] *Barker, R., Wichmann, B. Guidance for accredited laboratories on the use of computers. Accred Qual Assur 5, 287–288 (2000)*
[Online]. Available: <https://doi.org/10.1007/s007690000147>
- [3] *ISO 9000:2015 Quality management systems — Fundamentals and vocabulary, Sep. 2015.*
[Online]. Available: <https://www.iso.org/standard/45481.html>
- [4] *TraCIM Computational Aims Database: a Tutorial.* [Online]. Available: <https://www.tracim.eu/2123.html>
- [5] *ISO/IEC/IEEE 24765:2017(en), Systems and software engineering — Vocabulary.*
[Online]. Available: <https://www.computer.org/sevocab>
- [6] *ISO/IEC Guide 51:2014 Safety aspects — Guidelines for their inclusion in standards, 2014–04.* [Online]. Available: <https://www.iso.org/standard/53940.html>
- [7] *BS 10754-1:2018 Information technology — Systems trustworthiness, 2018.*
[Online]. Available: <https://knowledge.bsigroup.com/products/information-technology-systems-trustworthiness-governance-and-management-specification?version=standard>
- [8] *BIPM et al., ‘International vocabulary of metrology — Basic and general concepts and associated terms (VIM)’. [Online]. Available: https://www.bipm.org/documents/20126/2071204/JCGM_200_2012.pdf*
- [9] *European Metrology Network for Mathematics and Statistics: Quality Assurance Tools.* [Online]. Available: <https://www.euramet.org/european-metrology-networks/mathmet/activities/quality-assurance-tools>
- [10] *Lines K, Hippolyte J-L, George I, Harris P (2022). Acta IMEKO 11(4), 1–6. A MATHMET Quality Management System for data, software, and guidelines.* [Online]. Available: <https://doi.org/10.21014/actaimeko.v11i4.1348>
- [11] *The 61508 Association (2023). What is IEC 61508?*
[Online]. Available: <https://61508.org/knowledge/what-is-iec-61508/>
- [12] *Unified Modelling Language.* [Online]. Available: <https://www.uml.org/>
- [13] *SysML Open Source Project.* [Online]. Available: <https://sysml.org/>
- [14] *Semantic Versioning 2.0.0.* [Online]. Available: <https://semver.org/>
- [15] *Python Enhancement Proposals (PEPs), PEP 8 – Style Guide for Python Code.*
[Online]. Available: <https://peps.python.org/pep-0008/>
- [16] *Writing C (GNU Coding Standards).*
[Online]. Available: https://www.gnu.org/prep/standards/html_node/Writing-C.html
- [17] *Baker M (2016). Nature 533(7604), 452–454. 1500 scientists lift the lid on reproducibility.*
[Online]. Available: Nature Publishing Group. <https://doi.org/10.1038/533452a>
- [18] *EURAMET technical committees* [Online]. Available: <https://www.euramet.org/technical-committees>

APPENDIX I

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

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
Documentation of 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

Quality Requirement	SWIL 1	SWIL 2	SWIL 3	SWIL 4
Documentation of functional requirements	R	M	M	M
Traceability of requirements (user à functional à design à code à test)	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 Documentation of design	X	R	M	M
Review by team	R	M	M	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

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

APPENDIX II: LIFECYCLE

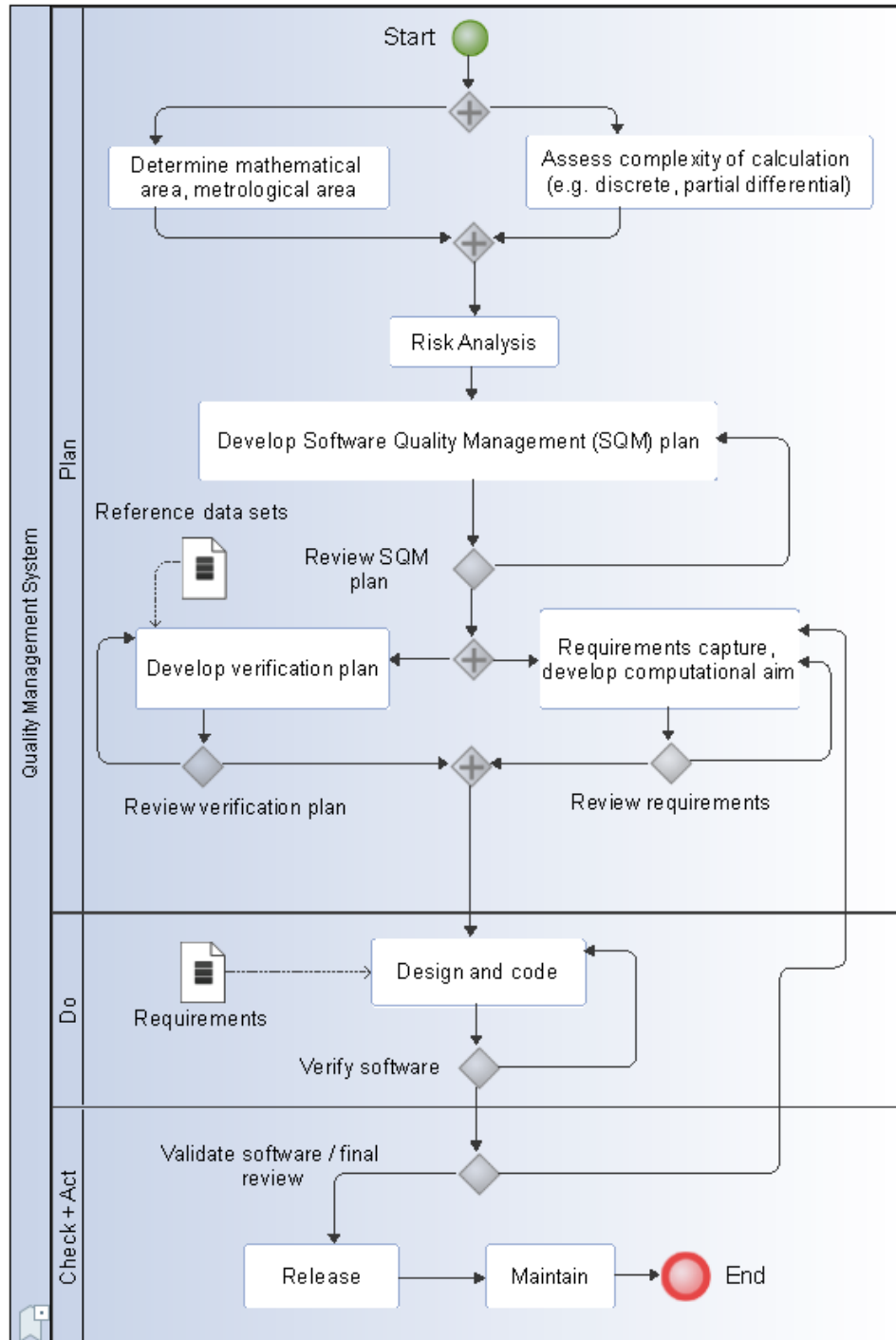


Figure 1: Example software lifecycle

Another possible lifecycle is illustrated in figure 1 of reference [10].

DOCUMENT HISTORY

Version	Date	Revised by	Change(s)	Approved by
1.0.0	July 2025	JLH, KL, IG	Initial live release.	NPL