A set of Common Software Quality Assurance Baseline Criteria for Research Projects

A DOI-citable version of this manuscript is available at http://hdl.handle.net/10261/160086.

This manuscript (permalink) was automatically generated from indigo-dc/sqa-baseline@1424d23 on January 18, 2021 with the use of https://gitlab.com/manubot/rootstock/.

Authors

- **Pablo Orviz**
  0000-0002-2473-6405 · orviz
  Spanish National Research Council (CSIC); Institute of Physics of Cantabria (IFCA)

- **Alvaro Lopez**
  0000-0002-0013-4602 · alvaro.lopez
  Spanish National Research Council (CSIC); Institute of Physics of Cantabria (IFCA)

- **Doina Cristina Duma**
  0000-0002-0124-4870 · caifti
  National Institute of Nuclear Physics (INFN)

- **Mario David**
  0000-0003-1802-5356 · mariojmdavid
  Laboratory of Instrumentation and Experimental Particle Physics (LIP)

- **Jorge Gomes**
  0000-0002-9142-2596 · jorge-lip
  Laboratory of Instrumentation and Experimental Particle Physics (LIP)

- **Giacinto Donvito**
  0000-0002-0628-1080
  National Institute of Nuclear Physics (INFN)

Abstract

The purpose of this document is to define a set of quality standards, procedures and best practices to conform a Software Quality Assurance plan to serve as a reference within the European research ecosystem related projects for the adequate development and timely delivery of software products.

Acknowledgements

The INDIGO-DataCloud, DEEP-Hybrid-DataCloud, eXtreme-DataCloud and EOSC-Synergy projects have received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 653549, 777435, 777367 and 857647 respectively.

This manuscript is a template (aka "rootstock") for Manubot, a tool for writing scholarly manuscripts. Use this template as a starting point for your manuscript.

The rest of this document is a full list of formatting elements/features supported by Manubot. Compare the input (`.md` files in the `/content` directory) to the output you see below.

Basic formatting

Bold text

Semi-bold text

Centered text

Right-aligned text

Italic text

Combined *italics and* ***bold***

~~Strikethrough~~

1. Ordered list item
2. Ordered list item
    a. Sub-item

    b. Sub-item
        i. Sub-sub-item
3. Ordered list item
    a. Sub-item

- List item
- List item
- List item

subscript: H_2O is a liquid

superscript: 2^{10} is 1024.

unicode superscripts^{0123456789}

unicode subscripts$_{0123456789}$

A long paragraph of text. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Putting each sentence on its own line has numerous benefits with regard to editing and version control.

Line break without starting a new paragraph by putting
two spaces at end of line.

Document organization

Document section headings:

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

A heading centered on its own printed page

Horizontal rule:

`Heading 1`'s are recommended to be reserved for the title of the manuscript.

`Heading 2`'s are recommended for broad sections such as *Abstract*, *Methods*, *Conclusion*, etc.

`Heading 3`'s and `Heading 4`'s are recommended for sub-sections.

Links

Bare URL link: https://manubot.org

Long link with lots of words and stuff and junk and bleep and blah and stuff and other stuff and more stuff yeah

Link with text

Link with hover text

Link by reference

Citations

Citation by DOI [1].

Citation by PubMed Central ID [2].

Citation by PubMed ID [3].

Citation by Wikidata ID [4].

Citation by ISBN [5].

Citation by URL [6].

Citation by alias [7].

Multiple citations can be put inside the same set of brackets [1,5,7]. Manubot plugins provide easier, more convenient visualization of and navigation between citations [2,3,7,8].

Citation tags (i.e. aliases) can be defined in their own paragraphs using Markdown's reference link syntax:

Referencing figures, tables, equations

Figure 1

Figure 2

Figure 3

Figure 4

Table 1

Equation 1

Equation 2

Quotes and code

> Quoted text

> Quoted block of text
>
> Two roads diverged in a wood, and I—
> I took the one less traveled by,
> And that has made all the difference.

Code `in the middle` of normal text, aka `inline code`.

Code block with Python syntax highlighting:

```
from manubot.cite.doi import expand_short_doi

def test_expand_short_doi():
    doi = expand_short_doi("10/c3bp")
    # a string too long to fit within page:
    assert doi == "10.25313/2524-2695-2018-3-vliyanie-enhansera-copia-i-
        insulyatora-gypsy-na-sintez-ernk-modifikatsii-hromatina-i-
        svyazyvanie-insulyatornyh-belkov-vtransfetsirovannyh-geneticheskih-
        konstruktsiyah"
```

Code block with no syntax highlighting:

```
Exporting HTML manuscript
Exporting DOCX manuscript
Exporting PDF manuscript
```

Figures

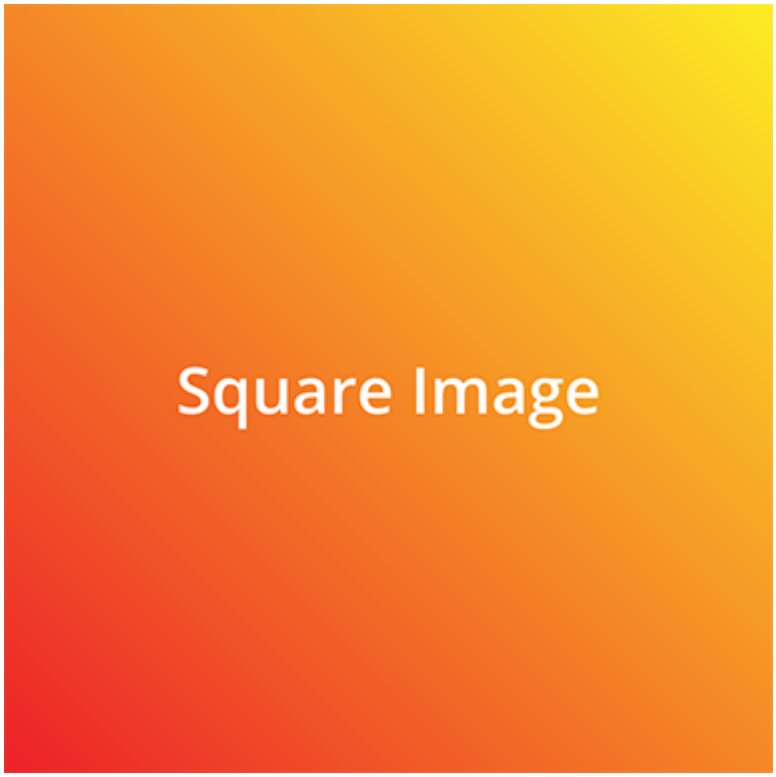

Figure 1: A square image at actual size and with a bottom caption. Loaded from the latest version of image on GitHub.

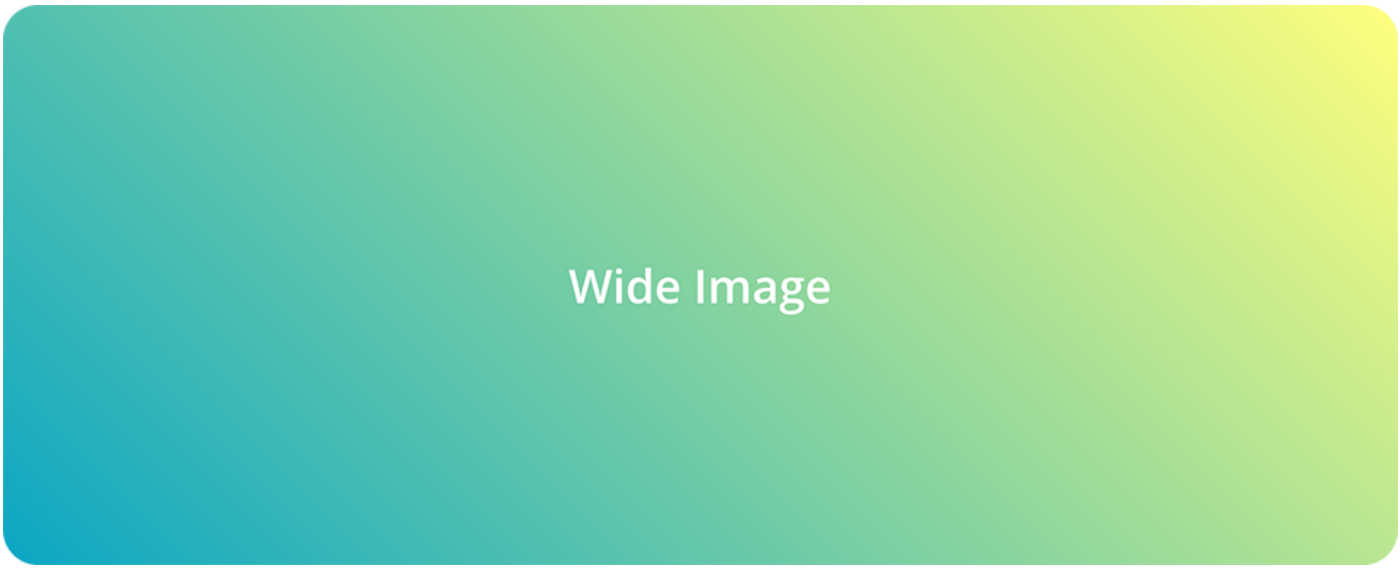

Figure 2: An image too wide to fit within page at full size. Loaded from a specific (hashed) version of the image on GitHub.

Figure 3: A tall image with a specified height. Loaded from a specific (hashed) version of the image on GitHub.

Figure 4: A vector `.svg` image loaded from GitHub. The parameter `sanitize=true` is necessary to properly load SVGs hosted via GitHub URLs. White background specified to serve as a backdrop for transparent sections of the image.

Tables

Table 1: A table with a top caption and specified relative column widths.

Bowling Scores	**Jane**	**John**	**Alice**	**Bob**
Game 1	150	187	210	105
Game 2	98	202	197	102
Game 3	123	180	238	134

Table 2: A table too wide to fit within page.

	Digits 1-33	**Digits 34-66**	**Digits 67-99**	**Ref.**
pi	3.14159265358979323 846264338327950	28841971693993751 0582097494459230	78164062862089986 2803482534211706	`piday.org`
e	2.71828182845904523 536028747135266	24977572470936999 5957496696762772	40766303535475945 7138217852516642	`nasa.gov`

Table 3: A table with merged cells using the `attributes` plugin.

	Colors	
Size	**Text Color**	**Background Color**
big	blue	orange
small	black	white

Equations

A LaTeX equation:

$$\int_0^\infty e^{-x^2} dx = \frac{\sqrt{\pi}}{2} \tag{1}$$

An equation too long to fit within page:

$$x = a + b + c + d + e + f + g + h + i + j + k + l + m + n + o + p + q + r + s + t + u + v + w + x + y + z + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 \tag{2}$$

Special

⚠ **WARNING** *The following features are only supported and intended for* `.html` *and* `.pdf` *exports. Journals are not likely to support them, and they may not display correctly when converted to other formats such as* `.docx`*.*

LINK STYLED AS A BUTTON

Adding arbitrary HTML attributes to an element using Pandoc's attribute syntax:

Manubot Manubot Manubot Manubot Manubot. Manubot Manubot Manubot Manubot. Manubot Manubot Manubot. Manubot Manubot. Manubot.

Adding arbitrary HTML attributes to an element with the Manubot `attributes` plugin (more flexible than Pandoc's method in terms of which elements you can add attributes to):

Manubot Manubot Manubot Manubot Manubot. Manubot Manubot Manubot Manubot. Manubot Manubot Manubot. Manubot Manubot. Manubot.

Available background colors for text, images, code, banners, etc:

`white` `lightgrey` `grey` `darkgrey` `black` `lightred` `lightyellow` `lightgreen` `lightblue` `lightpurple` `red` `orange` `yellow` `green` `blue` `purple`

Using the Font Awesome icon set:

✔ ? ★ 🔔 ⊗ •••

Light Grey Banner
useful for *general information* - manubot.org

Blue Banner
useful for *important information* - manubot.org

Light Red Banner
useful for *warnings* - manubot.org

Document Log

Issue	Date	Comment
v1.0	31/01/2018	First draft version
v2.0	05/02/2018	Updated criteria
v3.0	20/12/2019	Code management section, metadata for software
v3.1	05/03/2020	Add tags/names for each criteria
v3.2	23/04/2020	Add EOSC-Synergy to copyright
v3.3	15/10/2020	Fix issues: #32, #46, #47, #48, #49, #51

Introduction and Purpose

This document has been tailored upon the recommendations and requirements found in the Initial Plan for Software Management and Pilot Services deliverable [9], produced by the INDIGO-DataCloud project. These guidelines evolved throughout the project's lifetime and are being extended in the EOSC-Synergy, DEEP-Hybrid-DataCloud and eXtreme DataCloud subsequent projects. The result is a consolidated Software Quality Assurance (SQA) baseline criteria emanated from the European Open Science Cloud (EOSC), which aims to outline the SQA principles to be considered in the upcoming software development efforts within the European research community, and continuously evolve in order to be aligned with future software engineering practices and security recommendations.

Goals

1. Set the base of minimum SQA criteria that a software developed within EOSC development project MUST fulfill.
2. Enhance the visibility, accessibility and distribution of the produced source code through the alignment to the Open Source Definition [10].
3. Promote code style standards to deliver good quality source code emphasizing its readability and reusability.
4. Improve the quality and reliability of software by covering different testing methods at development and pre-production stages.
5. Propose a change-based driven scenario where all new updates in the source code are continuously validated by the automated execution of the relevant tests.
6. Adopt an agile approach to effectively produce timely and audience-specific documentation.

7. Lower the barriers of software adoption by delivering quality documentation and the utilization of automated deployment solutions.
8. Encourage secure coding practices and security static analysis at the development phase while providing recommendations on external security assessment.

Notational Conventions

The keywords “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 [11].

Quality Criteria

The following sections describe the quality conventions and best practices that apply to the development phase of a software component within the EOSC ecosystem. These guidelines ruled the software development process of the former European Commission-funded project INDIGO-DataCloud, where they have proved valuable for improving the reliability of software produced in the scientific European arena.

The next sections describe the development process driven by a change-based strategy, followed by a continuous integration approach. Changes in the source code, trigger automated builds to analyze the new contributions in order to validate them before being added to the software component code base. Consequently, software components are more eligible for being deployed in production infrastructures, reducing the likelihood of service disruption.

Code Accessibility [QC.Acc]

- **[QC.Acc01]** Following the open-source model, the source code being produced MUST be open and publicly available to promote the adoption and augment the visibility of the software developments.
- **[QC.Acc02]** Source code MUST use a Version Control System (VCS).
  - **[QC.Acc02.1]** It is RECOMMENDED that all software components delivered by the same project agree on a common VCS.
- **[QC.Acc03]** Source code produced within the scope of a broader development project SHOULD reside in a common organization of a version control repository hosting service.

Licensing [QC.Lic]

- **[QC.Lic01]** As open-source software, source code MUST adhere to an open-source license to be freely used, modified and distributed by others.
  - **[QC.Lic01.1]** Non-licensed software is exclusive copyright by default.
- **[QC.Lic02]** License MUST be compliant with the Open Source Definition [10].
  - **[QC.Lic02.1]** RECOMMENDED licenses are listed in the Open Source Initiative portal under the Popular Licenses category [12].
- **[QC.Lic03]** Licenses MUST be physically present (e.g. as a LICENSE file) in the root of all the source code repositories related to the software component.

Code Workflow [QC.Wor]

A change-based approach is accomplished with a branching model.

- **[QC.Wor01]** The main branch in the source code repository MUST maintain a working state version of the software component.
  - **[QC.Wor01.1]** Main branch SHOULD be protected to disallow force pushing, thus preventing untested and unreviewed source code from entering the production-ready version.
  - **[QC.Wor01.2]** New features SHOULD only be merged in the main branch whenever the SQA criteria is fulfilled.
- **[QC.Wor02]** New changes in the source code MUST be placed in individual branches.
  - **[QC.Wor02.1]** It is RECOMMENDED to agree on a branch nomenclature, usually by prefixing, to differentiate change types (e.g. feature, release, fix).
- **[QC.Wor03]** The existence of a secondary long-term branch that contains the changes for the next release is RECOMMENDED.
  - **[QC.Wor03.1]** Next release changes come from the individual branches.
  - **[QC.Wor03.2]** Once ready for release, changes in the secondary long-term branch are merged into the main branch and versioned. At that point in time, main and secondary branches are aligned. This step SHOULD mark a production release.
- **[QC.Wor04]** Semantic Versioning [13] specification is RECOMMENDED for tagging the production releases.

Code Management [QC.Man]

- **[QC.Man01]** An issue tracking system facilitates structured software development. Leveraging issues to track down both new enhancements and defects (bugs or documentation typos) is RECOMMENDED.
  - **[QC.Man01.1]** In addition to monitoring the internal development, issues are the best means for supporting users. External users SHOULD be able to create incidences based on the operational performance of the software.
  - **[QC.Man01.2]** The description of an issue SHOULD be concise and state clearly the problem. It is RECOMMENDED to add any reference to the actual problem. In the case of bugs, the issue SHOULD be accompanied by the relevant debug information. * The usage of templates for the issue description is RECOMMENDED.
- **[QC.Man02]** In social coding environments, pull requests represent the cornerstone of collaboration. A pull request provides a place for review and discussion of the changes proposed to be part of an existing version of the code.
  - **[QC.Man02.1]** Pull requests SHOULD be used for every change in the codebase.
  - **[QC.Man02.2]** A software project SHOULD be open to external collaboration through pull requests.
  - **[QC.Man02.3]** A pull request description SHOULD be concise and state clearly its purpose (e.g. if it is fixing an observed bug or adding a new feature)
    - The usage of templates for the pull request's description is RECOMMENDED.
  - **[QC.Man02.4]** It is RECOMMENDED to use pull requests to address open issues. The pull request description SHOULD make reference to the identifiers of the issues it is fixing (to eventually close them, either manually or automatically).

Code Style [QC.Sty]

Code style requirements pursue the correct maintenance of the source code by the common agreement of a series of style conventions. These vary based on the programming language being used.

- **[QC.Sty01]** Each individual software product MUST comply with community-driven or de-facto code style standards for the programming languages being used.
  - **[QC.Sty01.1]** Compliance with multiple complementary standards MAY exist.

- **[QC.Sty02]** Custom code style guidelines SHOULD be avoided, only considered in the hypothetical event of programming languages without existing community style standards.
  - **[QC.Sty02.1]** Custom styles MUST be documented by defining each convention and its expected output.
  - **[QC.Sty02.2]** Custom styles SHOULD evolve over time towards a more consistent definition.
- **[QC.Sty03]** Exceptions of individual conventions from the main definition are allowed but SHOULD bavoided
  - **[QC.Sty03.1]** Absence of standard conventions SHOULD be justified and tracked.
- **[QC.Sty04]** Code style compliance testing MUST be automated and MUST be triggered for each candidate change in the source code.

Code metadata [QC.Met]

Metadata for the software component provides a way to achieve its full identification, thus making software citation viable [14]. It allows the assignement of a Digital Object Identifier (DOI) and is key towards preservation, discovery, reuse, and attribution of the software component.

- **[QC.Met01]** A metadata file SHOULD exist along side the code, under its VCS. The metadata file SHOULD be updated when needed, as is the case of a new version.

Unit Testing [QC.Uni]

Unit testing evaluates all the possible flows in the internal design of the code, so that its behaviour becomes apparent. It is a key type of testing for early detection of failures in the development cycle.

- **[QC.Uni01]** Minimum acceptable code coverage threshold SHOULD be 70%.
  - **[QC.Uni01.1]** Unit testing coverage SHOULD be higher for those sections of the code identified as critical by the developers, such as units part of a security module.
  - **[QC.Uni01.2]** Unit testing coverage MAY be lower for external libraries or pieces of code not maintained within the product's code base.
- **[QC.Uni02]** Units SHOULD reside in the repository code base but separated from the main code.
- **[QC.Uni03]** Unit testing coverage MUST be checked on change basis.
- **[QC.Uni04]** Unit testing coverage MUST be automated.
  - **[QC.Uni04.1]** When working on automated testing, the use of testing doubles is RECOMMENDED to mimic a simplistic behaviour of objects and procedures.

Integration Testing [QC.Int]

Integration testing refers to the evaluation of the interactions among coupled software components or parts of a system that cooperate to achieve a given functionality.

- **[QC.Int01]** Integration testing outcome MUST guarantee the overall operation of the software component whenever new functionality are involved.
- **[QC.Int02]** Integration testing MAY be complemented with Acceptance and Scalability testing.
- **[QC.Int03]** Integration testing MUST NOT rely on non-operational or out-of-the-warranty services.
  - **[QC.Int03.1]** On lack of automation, pilot service infrastructures or local testbeds MAY be used.
  - **[QC.Int03.2]** In the presence of CI environments, integration tests SHOULD be suitable for the automated testing.
- **[QC.Int04]** Ad-hoc pilot service infrastructures and/or local testbeds MAY be used to cope with the integration testing requirements.
- **[QC.Int05]** Integration testing MAY NOT be viable to be checked on change basis, as it is likely to involve complex scenarios.

Functional Testing [QC.Fun]

Functional testing involves the verification of the software component's identified functionality, based on requested requirements and agreed design specifications. This type of software testing focus on the evaluation of the functionality that the software component exposes, leaving apart any internal design analysis or side-effects to external systems.

- **[QC.Fun01]** Functional testing SHOULD tend to cover the full set of functionality that the software component claims to provide.
- **[QC.Fun02]** Functional tests SHOULD be checked automatically with the exception of those functionality that require human interaction, such as Graphical User Interfaces (GUI).
- **[QC.Fun03]** When working on automated testing, the use of testing doubles is RECOMMENDED to mimic a simplistic behaviour of objects and procedures.
- **[QC.Fun04]** Functional tests SHOULD reside in the software component repository code base but separated from the main code.
- **[QC.Fun05]** Regression testing, that checks the conformance with previous tests, is covered at this stage by executing the complete set of functional tests available.
- **[QC.Fun06]** Functional and regression testing MUST be checked on change basis.
- **[QC.Fun07]** Functional and regression testing coverage MAY NOT be suitable for automated testing.

Test-Driven Development (TDD)

Test-Driven Development [15], is a software development process relying on software requirements being converted to test cases before software is fully developed, and tracking all software development by repeatedly testing the software against all test cases. This is opposed to software being developed first and test cases created later.

- **[QC.Tdd01]** Software requirements SHOULD be converted to test cases, and these test cases SHOULD be checked automatically.

Documentation [QC.Doc]

- **[QC.Doc01]** Documentation MUST be treated as code.
  - **[QC.Doc01.1]** Version controlled, it MAY reside in the same repository where the source code lies.
- **[QC.Doc02]** Documentation MUST use plain text format using a markup language, such as Markdown or reStructuredText.
  - **[QC.Doc02.1]** It is RECOMMENDED that all software components delivered by the same project agree on a common markup language.
- **[QC.Doc03]** Documentation MUST be online available in a documentation repository.
  - **[QC.Doc03.1]** Documentation SHOULD be rendered automatically.
- **[QC.Doc04]** Documentation MUST be updated on new software versions involving any substantial or minimal change in the behaviour of the application.
- **[QC.Doc05]** Documentation MUST be updated whenever reported as inaccurate or unclear.
- **[QC.Doc06]** Documentation MUST be produced according to the target audience, varying according to the software component specification. The identified types of documentation and their RECOMMENDED content are:
  - **[QC.Doc06.1]** README file
    - One-paragraph description of the application.
    - A "Getting Started" step-by-step description on how to get a development environment running (prerequisites, installation).
    - Automated test execution how-to.

        - Links to the external documentation below (production deployment, user guides).
        - Contributing code of conduct (optionally linked to an external CONTRIBUTING file)
        - Versioning specification.
        - Author list and contacts.
        - License information, with a link to the detailed description in an external LICENSE file.
        - Acknowledgments.
    - **[QC.Doc06.2]** Developer
        - Private API documentation.
        - Structure and interfaces.
        - Build documentation.
    - **[QC.Doc06.3]** Deployment and Administration
        - Installation and configuration guides.
        - Service Reference Card, with the following RECOMMENDED content:
        - Brief functional description.
        - List of processes or daemons.
        - Init scripts and options.
        - List of configuration files, location and example or template.
        - Log files location and other useful audit information.
        - List of ports.
        - Service state information.
        - List of cron jobs.
        - Security information.
        - FAQs and troubleshooting.
    - **[QC.Doc06.4]** User
        - Public API documentation.
        - Command-line (CLI) reference.
- **[QC.Doc07]** Documentation MUST be checked on change basis.

Security [QC.Sec]

- **[QC.Sec01]** Secure coding practices SHALL be applied into all the stages of a software component development lifecycle.
    - **[QC.Sec01.1]** Compliance with Open Web Application Security Project (OWASP) secure coding guidelines [16] is RECOMMENDED, even for non-web applications.
- **[QC.Sec02]** Source code SHALL use automated linter tools to perform static application security testing (SAST) [17] that flag common suspicious constructs that may cause a bug or lead to a security risk (e.g. inconsistent data structure sizes or unused resources).
- **[QC.Sec03]** Dynamic application security testing (DAST) [18] SHALL be performed from the outside, to software components in an operating state, to look for security vulnerabilities (e.g. SQL injection, cross-site scripting).
- **[QC.Sec04]** Interactive Application Security Testing (IAST) [19], analyzes code for security vulnerabilities while the app is run by an automated test. IAST SHOULD be performed to software components in an operating state, while Functional Testing **[QC.Fun]**, is running.
- **[QC.Sec05]** Manual penetration testing MAY be part of the application security verification effort.
- **[QC.Sec06]** Security code reviews [20] for certain vulnerabilities SHOULD be done as part of the identification of potential security flaws in the code. Inputs SHOULD come from automated linters and manual penetration testing results.
- **[QC.Sec07]** World-writable files or directories MUST NOT be present in the product's configuration or logging locations.
- **[QC.Sec08]** World-writable files SHOULD NOT be created while the service is in operation. Whenever they are required, the relevant files MUST be documented.
- **[QC.Sec09]** World-readable files MUST NOT contain passwords.
- **[QC.Sec10]** The services delivered SHALL adhere to any extra security policies or requirements set at the operational level.

Code Review [QC.Rev]

Code review implies the informal, non-automated, peer, human-based revision of any change in the source code [20]. It appears as the last step in the change management pipeline, once the candidate change has successfully passed over the required set of change-based tests.

- **[QC.Rev01]** Code reviews MUST be done in the agreed peer review tool within the project, with the following RECOMMENDED functionality:
  - **[QC.Rev01.1]** Allows general and specific comments on the line or lines that need to be reviewed.
  - **[QC.Rev01.2]** Shows the results of the required change-based test executions.
  - **[QC.Rev01.3]** Allows to prevent merges of the candidate change whenever not all the required tests are successful. Exceptions to this rule cover the third-party or upstream contributions which MAY use the existing mechanisms or tools for code review provided by the target software project. This exception MUST only be allowed whenever the external revision lifecycle does not interfere with the project deadlines.
- **[QC.Rev02]** Code reviews MUST be open and collaborative, allowing external expert revisions.
- **[QC.Rev03]** Code reviews SHOULD be concise and use neutral language. The areas where the reviewers MAY focus are:
  - **[QC.Rev03.1]** Message description: commit message is clear, self-explanatory and describes precisely the objectives being addressed.
  - **[QC.Rev03.2]** Goal or scope: change is needed and/or addresses/fixes the whole set of objectives.
  - **[QC.Rev03.3]** Code analysis: useless statements in the code, library or modules imported but never used or code style suggestions.
  - **[QC.Rev03.4]** Review of required tests: current battery of tests is sufficient for validation.
  - **[QC.Rev03.5]** Review of documentation: whether the change SHOULD bring along a corresponding update in the documentation.
- **[QC.Rev04]** Code reviews MUST be checked on change basis.
- **[QC.Rev05]** Code reviews SHOULD assess the inherent security risk of the changes, ensuring that the security model has not been downgraded or compromised by the changes.

Automated Deployment [QC.Aud]

- **[QC.Aud01]** Production-ready code SHALL be deployed as a workable system with the minimal user or system administrator interaction leveraging software configuration management (SCM) tools.
- **[QC.Aud02]** A SCM module is treated as code.
  - **[QC.Aud02.1]** Version controlled, it SHOULD reside in a different repository than the source code to facilitate the distribution.
- **[QC.Aud03]** It is RECOMMENDED that all software components delivered by the same project agree on a common SCM tool.
  - **[QC.Aud03.1]** However, software products are not restricted to provide a unique solution for the automated deployment.
- **[QC.Aud04]** Any change affecting the application's deployment or operation MUST be subsequently reflected in the relevant SCM modules.
- **[QC.Aud05]** Official repositories provided by the manufacturer SHOULD be used to host the SCM modules, thus augmenting the visibility and promote external collaboration.

Glossary

API
: Application Programming Interface

CLI
: Command Line Interface

DAST
: Dynamic Application Security Testing

EOSC
: European Open Science Cloud

IAST
: Interactive Application Security Testing

OWASP
: Open Web Application Security Project

SAST
: Static Application Security Testing

SCM
: Software Configuration Management

SQA
: Software Quality Assurance

TDD
: Test-Driven Development

VCS
: Version Control System

References

1. **Sci-Hub provides access to nearly all scholarly literature**
   Daniel S Himmelstein, Ariel Rodriguez Romero, Jacob G Levernier, Thomas Anthony Munro, Stephen Reid McLaughlin, Bastian Greshake Tzovaras, Casey S Greene
   eLife (2018-03-01) https://doi.org/ckcj
   DOI: 10.7554/elife.32822 · PMID: 29424689 · PMCID: PMC5832410

2. **Reproducibility of computational workflows is automated using continuous analysis**
   Brett K Beaulieu-Jones, Casey S Greene
   Nature biotechnology (2017-04) https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6103790/
   DOI: 10.1038/nbt.3780 · PMID: 28288103 · PMCID: PMC6103790

3. **Bitcoin for the biological literature.**
   Douglas Heaven
   Nature (2019-02) https://www.ncbi.nlm.nih.gov/pubmed/30718888
   DOI: 10.1038/d41586-019-00447-9 · PMID: 30718888

4. **Plan S: Accelerating the transition to full and immediate Open Access to scientific publications**
   cOAlition S
   (2018-09-04) https://www.wikidata.org/wiki/Q56458321

5. **Open access**
   Peter Suber
   MIT Press (2012)
   ISBN: 9780262517638

6. **Open collaborative writing with Manubot**
   Daniel S. Himmelstein, Vincent Rubinetti, David R. Slochower, Dongbo Hu, Venkat S. Malladi, Casey S. Greene, Anthony Gitter
   Manubot (2020-05-25) https://greenelab.github.io/meta-review/

7. **Opportunities and obstacles for deep learning in biology and medicine**
   Travers Ching, Daniel S. Himmelstein, Brett K. Beaulieu-Jones, Alexandr A. Kalinin, Brian T. Do, Gregory P. Way, Enrico Ferrero, Paul-Michael Agapow, Michael Zietz, Michael M. Hoffman, … Casey S. Greene
   Journal of The Royal Society Interface (2018-04-04) https://doi.org/gddkhn
   DOI: 10.1098/rsif.2017.0387 · PMID: 29618526 · PMCID: PMC5938574

8. **Open collaborative writing with Manubot**
   Daniel S. Himmelstein, Vincent Rubinetti, David R. Slochower, Dongbo Hu, Venkat S. Malladi, Casey S. Greene, Anthony Gitter
   PLOS Computational Biology (2019-06-24) https://doi.org/c7np
   DOI: 10.1371/journal.pcbi.1007128 · PMID: 31233491 · PMCID: PMC6611653

9. **INDIGO-DataCloud collaboration, Initial Plan for Software Management and Pilot Services**
   Members of the INDIGO-DataCloud collaboration
   (2015) https://owncloud.indigo-datacloud.eu/index.php/s/yDklCrWjKnjutVA

10. **The Open Source Definition, URL: https://opensource.org/osd**
Open Source Initiative
https://opensource.org/osd

11. **Key words for use in RFCs to Indicate Requirement Levels**
S. Bradner
(1997) https://www.rfc-editor.org/info/rfc2119

12. **Licenses & Standards, URL: https://opensource.org/licenses**
Open Source Initiative
https://opensource.org/licenses

13. **Semantic Versioning 2.0.0, URL: https://semver.org**
Tom Preston-Werner
https://semver.org

14. **Software citation principles**
Arfon M. Smith, Daniel S. Katz, Kyle E. Niemeyer, FORCE11 Software Citation Working Group
PeerJ Computer Science (2016-09-19) https://doi.org/bw3g
DOI: 10.7717/peerj-cs.86

15. **Test-driven development: by example**
Kent Beck
Addison-Wesley (2003)
ISBN: 9780321146533

16. **OWASP Secure Coding Practices-Quick Reference Guide** https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/migrated_content.html

17. **Source Code Analysis Tools | OWASP** https://owasp.org/www-community/Source_Code_Analysis_Tools

18. **Dynamic Application Security Testing, URL: https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools**
The OWASP Foundation
https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools

19. **What is IAST? Interactive Application Security Testing**
Veracode
https://www.veracode.com/security/interactive-application-security-testing-iast

20. **OWASP Code Review Guide** https://owasp.org/www-project-code-review-guide/migrated_content.html