CESSDA Software Maturity Levels (SMLs)

Description

This document was created so that readers interested in CESSDA Software Maturity Levels can find out more about the various qualities, levels, and requirements that underpin them.

Mandating and checking the interoperability or re/usability of the software components within CESSDA's technical Research Infrastructure is essential for promoting FAIRness in software products. However, there are inherent risks, such as the effort required for integration, maintenance of the technical framework, and adherence to required standards.

Document info

Status	Issued
Author(s)	John Shepherdson, Joshua Tetteh Ocansey
Date	27 June 2024
Licence	CC BY 4.0
Version	4.0
PID	10.5281/zenodo.12571107

Version history

Version	Release date	Comment	Revised by
4.0	June 2024	Revised to take account of containerisation.	John Shepherdson, Joshua Tetteh Ocansey
3.0	March 2019	Registered DOI via Zenodo.	John Shepherdson
2.1	N/A	Added minimum and expected levels for CA8.	John Shepherdson
2.0	December 2018	Following review by CESSDA CTO.	John Shepherdson
1.2	N/A	Moved to a new document template. Extensive changes to sections CA5 and CA6. Minor modifications to CA11. Added CA6. Minor modifications to CA11. Added section CA12. Now recognises the importance of TRLs wrt to EOSC.	John Shepherdson
1.1	N/A	Reviewed and updated all sections, including links. Global change from CRL to SML.	John Shepherdson
1.0	October 2017	Extracted from CESSDA Technical Architecture documentation.	John Shepherdson

Contents

Description **1**
Document info 1
Version history 2
Management Summary 4
Audience: 4
Purpose: 4
Software Maturity Levels **4**
Background 4
CESSDA Software Maturity Levels (SMLs) 6
CSQ 1: Documentation 6
CSQ1.1 End-user Documentation 6
CSQ1.2 Operational Documentation 6
CSQ1.3 Development Documentation 7
CSQ 2: Intellectual Property (IP) 7
CSQ 3: Extensibility 8
CSQ 4: Modularity 8
CSQ 5: Packaging 9
CSQ 6: Maintenance 9
CSQ 7: Verification and Testing 10
CSQ 8: Security (Privacy) **10**

Management Summary

Audience:

Potential providers of software artefacts for the CESSDA technical research infrastructure, including CESSDA Service Providers as the primary audience, with the possibility of engagement from any software development, including third parties suppliers. Additionally, research Infrastructures interested in adopting SML as a standard for their own infrastructure are welcome participants.

Purpose:

This document was created so that readers interested in CESSDA Software Maturity Levels can find out more about the various qualities, levels, and requirements that underpin them.

SML improves software FAIRness, which is not only a political imperative of European Research Infrastructure Consortiums' need to maximise their return on investment, but is also essential for growth amidst budget constraints, the need for re/usability and interoperability.

Mandating and checking the interoperability or re/usability of the software components within CESSDA's technical Research Infrastructure is essential for promoting FAIRness in software products. However, there are inherent risks, such as the effort required for integration, maintenance of the technical framework, and adherence to required standards.

Therefore, the need to measure the maturity of software designed for use by CESSDA is essential to upholding the quality of the technical research infrastructure.

Software Maturity Levels

Background

There are several methods to measure maturity. Capability Maturity Modelling (CMM) is used to assess services within Service Management frameworks such as FitSM or ITIL. Another method commonly used for assessing services is the Technology Readiness Levels (TRLs) scale. However, this does not address usability or reusability for software products, which is essential for the development of CESSDA's technical Research Infrastructure (RI).

Reuse Readiness Levels RRLs, which are developed by NASA Earth Science Data Systems, form the basis upon which the CESSDA software Maturity Level standards are made. RRLs address this gap in the assessment of the maturity of software artefacts.

The EU adopted TRLs as part of the H2020 programme, and both FitSM and TRLs have been subsequently adopted by the European Open Science Cloud (EOSC), which mandates that TRL Level 7 is the minimum acceptable for a tool or service to be included in their catalogue. Notably, both RRLs and TRLs were devised by and are widely used by NASA.

Table 1 shows the correspondence between the various levels in the RRL and SML scales. Given that one of CESSDA's goals is to have its tools and services listed in the EU Node Resource Hub, the requirements imposed by CESSDA will be continuously adjusted to guarantee compliance.

Reuse Readiness Levels	CESSDA Software maturity levels
• RRL1 - Limited reusability; not recommended for reuse • RRL2 - Initial reusability; reuse is not practical. • RRL3 - Basic reusability; might be reusable by skilled users at substantial effort, cost, and risk. RRL4 - Reuse is possible; it might be reused by most users with some effort, cost, and risk.	• SML1 - Use is feasible; the software can be used by skilled personnel, but with considerable effort, cost, and risk. assessment of effort, cost, and risk shall be made before use is attempted.
• RRL5 - Reuse is possible; might be reused by most users with some effort, cost, and risk. • RRL6 - Software is reusable; the software can be reused by most users, although there may be some cost and risk.	• SML2 - Use is possible for most users; with some effort, cost, and risk. A risk assessment should be made before use.
• RRL7 - Software is highly reusable; the software can be reused by most users at the minimum cost and risk. • RRL8 - demonstrated local reusability; the software has been reused by multiple users.	• SML3 - Demonstrable re/usability; There is clear evidence that the software is widely used by many users and use cases.

RRL9 - demonstrated extensive reusability; the software is being reused by many classes of users over a wide range of systems.	

CESSDA Software Maturity Levels (SMLs)

The SML guidance is assessed based on **Minimum**, **Expected**, and **Excellent** standards for each quality, known as CESSDA SML Qualities (CSQ). The standards are employed to assess the products delivered by software product providers. The CSQ ensures that each SML level meets at least 80% of its requirements. Additionally, it follows an inheritance approach to the SML standard; for example, all SML 2 requirements should inherit those from SML 1, and SML 3 inherits most requirements from SML 2.

CSQ 1: Documentation

CSQ1.1 End-user Documentation

SML1:

1. Web accessible user documentation.
2. Web accessible documentation for user interface

SML2:

3. Documentation must demonstrate the key functionalities.
4. Terminology is explained (as in a glossary).
5. The documentation is consistent with the version of the software.

SML3:

6. Includes a walk-through tutorial.
7. Contains examples of various use case customisations.
8. Documentation can be used for training.

CSQ1.2 Operational Documentation

SML1

1. Web accessible configuration documentation.

SML2:

2. Documentation is sufficient to gain an understanding of the deployment and configuration without technical support.
3. Terminology is explained (as in a glossary).
4. The documentation is consistent with the version of the software.
5. Exceptions and failure messages are explained.

SML3:

6. Example of walk-through tutorial.
7. Demonstrations of various configuration changes.
8. Upgrade workflow is fully documented.
9. Documentation can be used to deploy and configure software using different methods.
10. Documentation can be used for training.
11. Documentation has comments and feedback sections.

CSQ1.3 Development Documentation

SML1:

1. Documentation should explain the technologies used and their constraints.

SML2:

2. Web accessible documentation for API functionality.
3. Terminology is explained (as in a glossary).
4. The codebase, including modules, is well documented and commented on.
5. The documentation is consistent with the version of the software.

SML3:

6. Documentation on how to extend or add plug-ins to software.
7. Documentation has comments and feedback sections.
8. Documentation provides software development models
9. Documentation contains badges or links to software quality pages (for example, SonarQube, SQAaaS, GitLab, GitHub, and CodeSonar).

CSQ 2: Intellectual Property (IP)

SML1:

1. Software Developers are identified and documented.
2. Software Developers contacts and responsibilities are documented.

3. IP Statements contain recommended citations or recognised licences.

SML2:

4. Guidelines or policies of developers or developers' organisations are linked and can be reviewed.
5. Evidence of an IP rights agreement that would result from cooperative activities.
6. IP rights statements should have "limited right of use," a condition of use, or both commercial and non-commercial use.

SML3:

7. Developers can be contacted through a single point for formal statement, negotiations, etc.
8. There is machine-readable code for IP statements.

CSQ 3: Extensibility

SML1:

1. Software source code is available and accessible.
2. Logical flow of source code is documented and can be understood.

SML2:

3. There exists a use case (features) for software to be extended.
4. Evidence of some programming practices designed to enable extensibility.
5. Evidence of well-defined API for future extensibility

SML3

6. There is a defined procedure for extending the software.
7. There is evidence that software has been extended by external people or groups.
8. There is a library available for user-generated content for extension to multiple domains.
9. There can be user-generated documentation on extensions.

CSQ 4: Modularity

SML1:

1. Source code is organised into a primary system that provides functionality.
2. Code within each module contains many independent logical paths.

3. Internal functions or interfaces are accessible by external programs through the primary system.

SML2:

4. Distinction is made between general and specific use-case functionality.
5. Architecture is open and fully structured into individual components that provide functions or interfaces.

SML3:

6. There is consistent error handling with meaningful messages.
7. Utilises generic extensions in programming language to enhance type checking and compile-time error validation.
8. Evidence of software structure based on software models or business processes (UML, BPMN, etc.).

CSQ 5: Packaging

SML1:

1. Configuration and deployment information, or documentation, is available.
2. Configuration files are separated from the core software.
3. Software has a package.

SML2:

4. The software is containerised (e.g. with Docker).
5. Configuration information is available and can be scripted to deploy the packaged/containerised software from the command line.

SML3:

6. Versions of deployed software can be upgraded/rolled back automatically.
7. Software can be packaged into various forms (i.e., binary files, Docker, etc)

CSQ 6: Maintenance

SML1:

1. Developers (organisation) information, including contacts, are available.

SML2:

2. There is evidence that developers (organisations) respond to issues.
3. There is a developers' organisation page dedicated to issues, patches, updates, FAQ etc.

SML3:

4. There is evidence of a user community or discussion group.
5. There is evidence of regular updates or upgrades.
6. There is evidence of a roadmap for upgrade and additional feature development.

CSQ 7: Verification and Testing

SML1:

1. Software applications are buildable and runnable.

SML2:

2. There is evidence of unit testing.
3. There is evidence of API testing, if applicable.

SML3:

4. There is evidence of a stable release version of the application.
5. There is integration testing among different modules.
6. There is verification for different use cases.

CSQ 8: Security (Privacy)

SML1:

1. There is evidence of published privacy policies and how privacy is handled.

SML2:

2. Sensitive configuration information (such as user credentials) is encrypted.

SML3:

3. There is a mechanism to check software code security.
4. There is evidence that software code vulnerabilities are resolved.