Report on operational software maturity level methodology

CONTRACT NO	EESI2 312478
INSTRUMENT	CSA (Support and Collaborative Action)
THEMATIC	INFRASTRUCTURE

Due date of deliverable:	June 1st, 2013
Actual submission date:	August 10th, 2013
Updated revision 1.1:	March 1st, 2014
Updated revision 1.2:	July 1st, 2015

Duration: 30 months

Name of lead contractor for this deliverable: Bernd Mohr, PRACE-JSC

Name of reviewers for this deliverable: Francois Bodin (University of Rennes), Andrew Jones (NAG), Lee Margetts (University of Manchester)

Abstract: Report on operational software maturity level methodology: This report will document the methodology for estimating the level of maturity of Exascale software components developed.

Revision 1.2

Project co-funded by the European Commission within the Seventh Framework Programme (FP7/2007-2013)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Table of Contents

Glossary

Abbreviation / acronym	**Description**
HPC	High Performance Computing
OMM	Open Maturity Model
OSS	Open source software
TRL	Technology Readiness Level

1. Executive Summary

One of the important recommendations of the first phase of the European Exascale Software Initiative (EESI1) was to establish a European Exascale Software Centre to coordinate research, development, testing, and validation of EU HPC Exascale software ecosystem components and modules. As a first step in this direction, the main charter of work package 6 of EESI2 is to investigate the feasibility and prepare for the operation of such a centre. To accomplish this, the objectives of WP 6 are to:

- Develop and document a methodology for estimating the level of maturity of Exascale software components.
- Identify three software stack components from existing and near future European Exascale projects and apply the defined methodology.
- Examine existing “equivalent" centres and propose a structure adapted to Exascale software.

This deliverable (D6.1) reports on first results on these objectives as defined by WP6 Task 6.1 (“Evaluation methodology set up”).

First, we investigated existing maturity level methodologies in other technical areas and non-HPC software environments. Two important findings which influenced our discussions and our proposal for an HPC software maturity level model are various “Technology Readiness Level” (TRL) assessments used by various organisations in different technical fields (e.g., US DoD, US Air Force, NASA, European Space Agency) and the QualiPSo Open Maturity Model (OMM), one of the results of the EU FP6 project QualiPSo. These and other existing approaches are detailed in Chapter 2.

Second, in Chapter 3 we present a first proposal for a HPC software maturity level model based on discussions with HPC development experts inside and outside the EESI2 project. We define four main HPC software component classes, namely scientific application codes, libraries and frameworks, development tools, and programming model implementations, to clearly set the context for our proposal. The core of our model are the assessment criteria for HPC software maturity. These are documentation, support, availability, coverage, portability, scalability, performance, and quality.

2. Maturity level methodologies in other technical areas and non-HPC software environments

To our knowledge, there are no published methodologies for the assessment of the maturity of open-source HPC software. However, corresponding methodologies exist for a long time already for other technical areas and of course also for (traditional) software development in general. For example, there is an extensive series of standards by ISO/IEC called SQuaRE ("Software product Quality Requirements and Evaluation") with numbers of the form ISO/IEC 250mn, e,g, ISO/IEC 25010:2011.

In the following, we discuss some existing maturity level methodologies in other technical areas and non-HPC software environments. Two important findings which influenced our discussions and our proposal for an HPC software maturity level model are various "Technology Readiness Level" (TRL) assessments used by various organisations in different technical fields and the QualiPSo Open Maturity Model (OMM), one of the results of the EU FP6 project QualiPSo.

2.1 Technology Readiness Level (TRL)

Technology Readiness Level (TRL) is a measure used to assess the maturity of evolving technologies (devices, materials, components, software, work processes, etc.) during its development and in some cases during early operations. Generally speaking, when a new technology is first invented or conceptualized, it is not suitable for immediate application. Instead, new technologies are usually subject to experimentation, refinement, and increasingly realistic testing. Once the technology is sufficiently proven, it can be incorporated into a (production) system/subsystem.

Various organisations require the use of TRLs, and although they are conceptually similar, significant differences exist in terms of maturity at a given technology readiness level. The following TRLs are publically documented:

- U.S. Department of Defense (DoD), besides TRL also has a Software TRL
- National Aeronautics and Space Administration (NASA)
- U.S. Air Force
- European Space Agency (ESA)
- Oil and gas industry
- U.S. Federal Aviation Administration (FAA)
- U.S. States Department of Energy (DOE) uses Technology Readiness Assessments (TRAs) and developing Technology Maturation Plans (TMPs).

2.1.1 Assessment of TRLs

An important part of a TRL specification, besides defining the actual technology readiness levels for the target area, is the process and methodology which describes how the TRL is determined for a specific component or piece of technology. Different TRL specifications differ in how formal and precise the processes are defined. They range from informal, general descriptions, well-defined questionnaires, to partially or fully-automated procedures.

For example, a Technology Readiness Level Calculator was developed by the United States Air Force. This tool is based on a standard set of questions implemented in Microsoft Excel that produces a graphical display of the TRLs achieved. It is intended to provide a snapshot of technology maturity at a given point in time.

Another example is the Technology Program Management Model that was developed by the United States Army. The TPMM is a TRL-gated high-fidelity activity model that provides a flexible management tool to assist Technology Managers in planning, managing, and assessing their technologies for successful technology transition. The model provides a core set of activities including systems engineering and program management tasks that are tailored to the technology development and management goals. This approach is comprehensive, yet it consolidates the complex activities that are relevant to the development and transition of a specific technology program into one integrated model.

2.1.2 Example: European Space Agency

As an example, the following table shows the TRL used by ESA. Instruments and spacecraft sub-systems are classified according to a TRL on a scale of 1 to 9. Levels 1 to 4 relate to creative and innovative technologies before or during the mission assessment phase. Levels 5 to 9 relate to existing technologies and to missions in definition phase.

Technology Readiness Level	**Description**
TRL 1	Basic principles observed and reported
TRL 2	Technology concept and/or application formulated
TRL 3	Analytical & experimental critical function and/or characteristic proof-of-concept
TRL 4	Component and/or breadboard validation in laboratory environment
TRL 5	Component and/or breadboard validation in relevant environment
TRL 6	System/subsystem model or prototype demonstration in a relevant environment (ground or space)
TRL 7	System prototype demonstration in a space environment
TRL 8	Actual system completed and "Flight qualified" through test and demonstration (ground or space)
TRL 9	Actual system "Flight proven" through successful mission operations

2.1.3 Critical Discussion of TRLs

The primary purpose of using Technology Readiness Levels is to help management in making decisions concerning the development and transitioning of technology. It should be viewed as one of several tools that are needed to manage the progress of research and development activity within an organization.

Among the advantages of TRLs are:

- Provide a common understanding of technology status
- Risk management
- Used to make decisions concerning technology funding
- Used to make decisions concerning transition of technology
- Understood by policy makers

Some of the characteristics of TRLs that limit their utility:

- Readiness does not necessarily fit with appropriateness or technology maturity
- Numerous factors must be considered and these factors depend on the context
- Current TRL models tend to disregard negative and obsolescence factors. However, there have been suggestions made for incorporating such factors into assessments.

2.2 ISO/ENC 25010:2011

ISO/IEC 9126 “Software engineering - Product quality” was an international standard for the evaluation of software quality. It has been replaced by ISO/IEC 25010:2011. The quality model presented in the first part of the standard, classifies software quality in a structured set of characteristics and sub-characteristics as follows:

- **Functionality** - A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.
    - Suitability
    - Accuracy
    - Interoperability
    - Security
    - Functionality Compliance
- **Reliability** - A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.
    - Maturity
    - Fault Tolerance
    - Recoverability
    - Reliability Compliance
- **Usability** - A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.
    - Understandability
    - Learnability
    - Operability
    - Attractiveness
    - Usability Compliance
- **Efficiency** - A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.
    - Time Behaviour
    - Resource Utilization
    - Efficiency Compliance
- **Maintainability** - A set of attributes that bear on the effort needed to make specified modifications.
    - Analyzability
    - Changeability
    - Stability
    - Testability
    - Maintainability Compliance
- **Portability** - A set of attributes that bear on the ability of software to be transferred from one environment to another.
    - Adaptability
    - Installability
    - Co-Existence
    - Replaceability
    - Portability Compliance

Each quality sub-characteristic (e.g. adaptability) is further divided into attributes. An attribute is an entity which can be verified or measured in the software product. Attributes are not defined in the standard, as they vary between different software products.

Software product is defined in a broad sense: it encompasses executables, source code, architecture descriptions, and so on. As a result, the notion of user extends to operators as well as to programmers, which are users of components such as software libraries.

The standard provides a framework for organizations to define a quality model for a software product. On doing so, however, it leaves up to each organization the task of specifying precisely its own model. This may be done, for example, by specifying target values for quality metrics which evaluates the degree of presence of quality attributes.

2.3 QualiPSo Open Maturity Model (OMM)

The EU FP6-IST project “Quality platform for open source software” (QUALIPSO) ran from 2006 to 2010. The project website can be found at http://www.qualipso.org/. QualiPSo aimed at making a major contribution to the state of the art and practice of Open Source Software. The goal of the project was "to define and implement technologies, procedures and policies to leverage the Open Source Software development current practices to sound and well recognised and established industrial operations".

QualiPSo brought together software companies, application solution developers and research institutions and was driven by the need for having the appropriated level of trust for Open Source Software (OSS) which makes OSS development an industrial and widely accepted practice. To reach this goal the QualiPSo project defined, deployed and launched the QualiPSo Competence Centres in Europe (4), Brazil (1), and China (1) all of them making use of the QualiPSo Factory (a new generation source-forge).

OMM is a process model for OSS development by developers and integration of OSS components by integrators. OMM is organized in three levels, each level building on and including trustworthy elements at the lower level:

- Basic Level (measurable)
  - Production documentation
  - Use of established and widespread standards
  - Quality of test plan
  - Licenses
  - Technical environment
  - Number of commits and bug reports
  - Maintainability and stability
  - Configuration, requirements management
  - Project planning
  - Availability and use of a (product) roadmap
- Intermediate Level (partially measurable to subjective)
  - Project planning and monitoring
  - Design and test
  - Process and product integration
  - Relationship between stakeholders
- Advanced Level (subjective)
  - Product integration
  - Reputation
  - Results of third party assessment
  - Contribution to OSS product from S/W companies
  - Risk management

More information on the OMM and its assessment can be found at http://qualipso.icmc.usp.br/OMM/. The paper “Towards The Evaluation of OSS Trustworthiness: Lessons Learned From The Observation of Relevant OSS Projects” written by QualiPSo project members and presented at the IFIP 20th World Computer Congress, Working Group 2.3 on Open Source Software (OSS 2008) provides some experiences using OMM on existing OSS, see http://dx.doi.org/10.1007/978-0-387-09684-1_37.

2.4 The UK Software Sustainability Institute

The UK Software Sustainability Institute (SSI) is a UK National Facility based at the universities of Edinburgh, Manchester, Oxford, and Southampton (see http://www.software.ac.uk/).

The goal of the SSI is to ensure that software generated as an output of research is written and supported in a sustainable way. This means that where appropriate, it has been engineered in such a way that it is possible to improved and supported in the future. The SSI does this by working with researchers, developers, funders, and infrastructure providers to identify key issues and best practices surrounding scientific software.

Their processes involve assessing software and making recommendations as to how that software can be best managed as a sustainable resource. They have finite government funding to provide consultancy and software engineering effort. This effort is allocated through periodic competitions which are peer reviewed.

The SSI has adopted a 2-level model of software maturity assessment. In the first level, software developers can carry out a self-assessment for their software using an online tool. In the second level, a more in-depth assessment can be carried out either via an allocation of SSI effort (via the periodic competitions) or for a fee.

The 2-level model is attractive. The first level enables research groups to assess their own software and improve their "best practice" procedures as their software evolves. The second level gives an independent assessment of maturity (sustainability) useful to funding agencies, end user communities, and industry.

The SSI operates a software assessment model that is potentially transferable to the Exascale community.

2.5 Other open-source software assessment methodologies

Besides QualiPSO OMM, there are numerous other assessment methodologies for open-source software, among them:

- Open Source Maturity Model (OSMM) from Capgemini
- Open Source Maturity Model (OSMM) from Navica
- Methodology of Qualification and Selection of Open Source software (QSOS)
- Open Business Readiness Rating (OpenBRR)
- Open Business Quality Rating (OpenBQR)
- QualiPSo Model for Open Source Software Trustworthiness (MOSST)
- QualOSS – Quality of Open Source

However, to our knowledge, so far, none of the methodologies takes special requirements of HPC system software and applications into account yet (for example, non-standard hardware platforms or inherent parallel executions).

3. A proposal for a HPC software maturity level model

While many TRL specifications and other open-source software assessment methodologies exist as described in the previous chapter, they cannot be easily adapted or modified to apply them to HPC software components. The main reason is the cluster architecture of HPC systems: tests which examine the full functionality, scalability, and performance of the HPC components need to be executed on real-scale HPC systems (and cannot be realistically performed on simple server platforms used for development). The diversity and non-standard methods of job launching mechanisms on HPC clusters complicates this further. Finally, while many codes use “industry-standard” parallel programming models like MPI or OpenMP, many other do not and deploy non-standard or newly-proposed (not widespread) models.

Therefore, the first part of the work agenda of the EESI2 project work package 6, called Task 6.1 (“Evaluation methodology set up”) is to

- Develop and document a concrete and precise methodology for estimating the level of maturity of Exascale software components including:
    - Which tests should be performed, wow should they be performed, and by whom
    - Where tests should be performed: centralized or distributed, size, mission and objectives, competences
    - Definition of multiple descriptors for maturity estimation
    - Definition of how the outcomes of the tests and the maturity level should be documented
- Define what Exascale software components should be evaluated in Europe taking into account the developments of current and near future European Exascale projects.

The state of discussions, first results and outcomes of this task are described below, together with open issues which need further discussions and work.

3.1 HPC software component classes

The long-term goal of this effort is to define a working maturity level assessment procedure for all system, middleware, and application software components of an HPC system. However, some aspects of the assessment methodology, for example performance or scalability tests, strongly depend on the type of the software component. As the development resources and the duration of the EESI2 project as a support action are limited, it was decided - in the context of this report - to concentrate on a few key classes of HPC software components. These will be described in more detail below.

3.1.1 Application codes (AC)

For this report, we define an application code as a stand-alone scientific simulation program which has a clearly defined single purpose (e.g. simulation of molecular dynamics). It typically comes in the form of a single executable (or a set of interconnected executables for MPMD programs). If an HPC simulation program is implemented in a very flexible way and the methodologies, algorithms, solvers actually used in the course of the computation can be specified via input or configuration files, an instance of an application code for our assessment methodology is defined by the combination of the simulation program and a corresponding configuration/input file.

3.1.2 Libraries and Frameworks (LF)

In contrast to application codes, libraries and frameworks are building blocks which can be used for the development of application codes, for example mathematical libraries or solvers.

The distinguishing feature of Libraries and Frameworks is that they are software components which are building blocks mainly for application codes but also for development tools or programming model implementations. This means that they cannot be used (tested) as is but a (test) main program that exercises features of the library or framework is needed. A typical example is a numerical library.

3.1.3 Development Tools (DT)

Development tools are software components which can be applied to application codes and other software components, for example, to improve their quality or performance. This includes, among others, debuggers and validation tools, tools for performance measurement and analysis, performance modelling and prediction, static code analysers, and code quality assessment tools.

3.1.4 Programming model implementations (PM)

Programming model implementations are software components which implement and provide programming models for developing software. For HPC, this includes compilers for parallel programming languages and models like UPC, CAF, OpenMP, or OpenACC, and their associated run-time libraries, but also libraries for parallel programming like MPI or SHMEM. Programming model implementations are used to implement the basic algorithms and procedures of application codes, libraries and frameworks, and development tools.

3.2 A Possible TRL for Exascale Software

A possible TRL for HPC system and application software is shown in Figure 1. Levels 1 to 5 relate to the demonstration stage, levels 6 and 7 represent the prototype phase, and level 8 and 9 cover the final product stage.

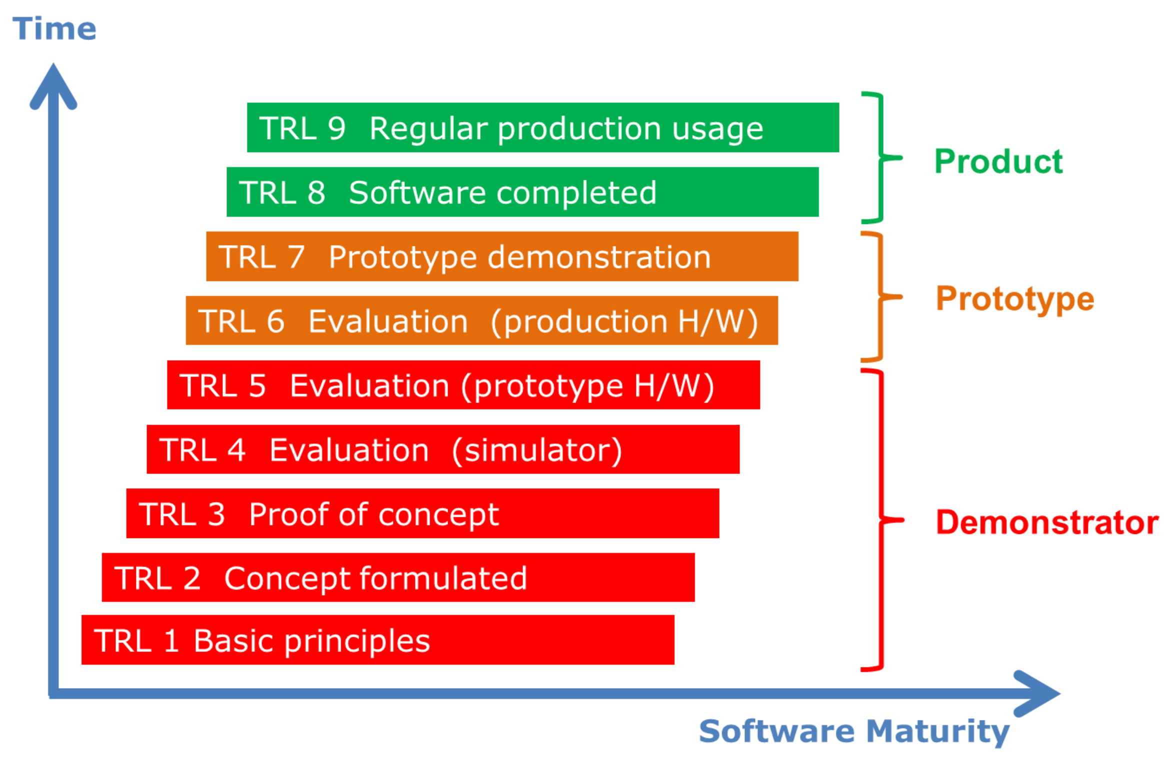

Time
TRL 9 Regular production usage
TRL 8 Software completed
TRL 7 Prototype demonstration
TRL 6 Evaluation (production H/W)
TRL 5 Evaluation (prototype H/W)
TRL 4 Evaluation (simulator)
TRL 3 Proof of concept
TRL 2 Concept formulated
TRL 1 Basic principles
Product
Prototype
Demonstrator
Software Maturity

Figure 1: Possible TRL for Exascale software

3.3 Assessment criteria for HPC software maturity

In this section we describe the main factors which we believe are indicators of the maturity level of HPC software components and therefore can be used as assessment criteria for HPC software maturity. They form the quality model as defined by ISO/ENC 25010:2011 (see Section 2.2). For each factor, we try to list directly measurable indicators or, if direct indicators do not exist, provide a list of "proxy" indicators which indicate the maturity.

3.3.1 Documentation and Support

- Availability of Documentation
    - Up-to-date installation guide, user guide, FAQ, quick introduction (getting started), release notes, etc.
    - Up-to-date training materials, manuals, guidelines
    - Up-to-date training classes or courses (also online like webinars or downloadable videos)
    - Availability of professional (commercial, contracted) training
    - Published books by persons not affiliated with the component
    - Documentation available in multiple languages
    - Published procedure for user (feature) requests
    - Product examples or demos
    - Published roadmap (future platforms and features)
- Support (bug fixing, help)
    - Website, support mailing list, user forum, phone service
    - 24/7 support or best effort
    - Usage of bug tracking and/or ticket systems
    - Availability of professional (commercial, contracted) support
    - Average bug time solving
    - Frequency of releases and patches
- Long-term commitment to
    - Further development
    - Maintenance
    - Support
    - Active maintainer organisation / sponsor

3.3.2 Availability and Coverage

- Licensing
    - Open-source, free licences, paid licences
- Distribution media
    - Online, CD/DVD
    - Source code or binary
    - Binary packages (e.g RPM)
    - Source-code based package installers (e.g. EasyBuild)
    - Access to project repository (read-only or completely open, e.g. GitHub) and/or ticket system
- Organisational form of (distributed) developer consortium
    - Open-source
        - Ad-hoc
        - Leader + (occasional) contributors
        - Well-defined governance model
    - Closed / commercial development
- Multiple sources (PM, LF only)
    - Multiple implementations available
    - Standards, especially multiple versions of a standard
- Coverage (LF, DT, PM only)

    - Languages (C, C++, Fortran, others?) and language versions (C99, Fortran 2003, etc)
    - Platforms beyond PRACE
    - Accelerator support (GPGPU, Intel MIC, …)
- Coverage (AC, PM only)
    - Application areas / use cases
- Number of users / installations / component downloads

3.3.3 Portability and Scalability

- Installation test on major PRACE platforms
    - BlueGene, Cray, Linux cluster, other?
- Execution test on platforms using a reasonable number of nodes/cores (1024 nodes?)
    - AC: with a specified set of input sets (small, medium, large), should include output correctness check
    - LF: with set of example or benchmark codes if provided by component
    - DT: should at least works with PRACE benchmark sets
    - PM: with set of example or benchmark codes if provided by component
- Scalability
    - Like portability execution tests but with more and more nodes/cores
    - Execution must be still efficient / performing

3.3.4 Performance and Quality

- Code quality
    - Coherent usage of coding guidelines/standard
    - Source code quality
    - Programming language uniformity
    - Existence of developer documentation
    - Use of (automated) standardized (regression) tests
- Ease-of-use
    - Detailed feature description and user manual
    - Existence of examples and demos
    - Ease of installation, configuration, and usage
- Testability
    - Build-in distribution and installation checks
    - Availability of test suites
    - Availability of benchmark suites, benchmark data sets or microbenchmarks
- Fault-tolerant / resilience support
    - Check-pointing/restart/
- Re-usability
- Efficiency
- User satisfaction
    - List of organisations, testimonials, and other projects using this software
    - Case studies, usage histories
    - Reports from forums, blogs, mailing lists, newsgroups, magazines, scientific articles
    - Organized user community (e.g. annual user group meetings or BoFs)
    - Quality of support services
        - Average response time
        - Average number of contacts with customer to resolve issue

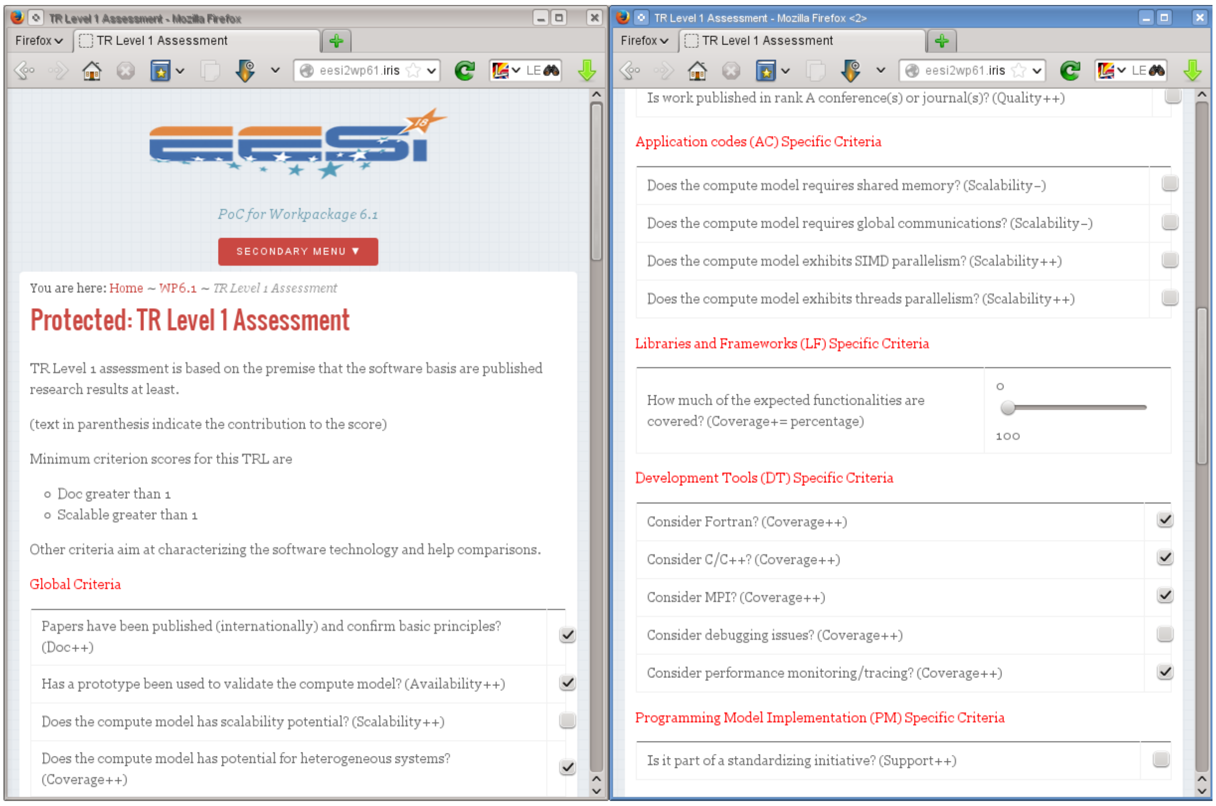
3.4 Open issues and next steps

Of course, the proposed HPC software maturity level model described in sections 3.1 to 3.3 is far from being complete and well-defined. More discussions, especially involving further experts from other relevant PRACE-3IP and PRACE-4IP work packages but also external experts are necessary to finalize the exact definition of the envisioned methodology.

The actual collection of the quality model factors, and their evaluation to determine the corresponding TRL could be done in various ways:

- The simplest form would be that interested projects (or external evaluators) document the quality model factor values in a simple document which follows the structure of Section 3.3 and documents the values and answers for each item in text form. A simple template document can be easily extracted from this document. Whenever possible, the source of the information should be documented as well (URLs or proper publication references).

  Then, based on the collected information and answers, the TRL is estimated for each of the major group of factors (Documentation and Support, Availability and Coverage, Portability and Scalability, and Performance and Quality). The overall TRL could then be determined by averaging the four sub values.
- Ideally, the input for the quality criteria factors would be collected with the help of a web-based online form, standardizing the input as much as possible, making it possibly to automatically evaluate the collected input (e.g. by scoring the answers) and the Exascale TRL could be automatically determined by creating a map between score ranges and TRLs. An "proof-of-concept" online form for a Exascale TRL Level 1 assessment is shown below in Figure 2:


TR Level 1 Assessment - Mozilla Firefox
Firefox
TR Level 1 Assessment
eesi2wp61.iris
TR Level 1 Assessment - Mozilla Firefox <2>
PoC for Workpackage 6.1
SECONDARY MENU ▼

You are here: Home ~ WP6.1 ~ TR Level 1 Assessment

Protected: TR Level 1 Assessment

TR Level 1 assessment is based on the premise that the software basis are published research results at least.

(text in parenthesis indicate the contribution to the score)

Minimum criterion scores for this TRL are

- Doc greater than 1
- Scalable greater than 1

Other criteria aim at characterizing the software technology and help comparisons.

Global Criteria

Papers have been published (internationally) and confirm basic principles? (Doc++)	✓
Has a prototype been used to validate the compute model? (Availability++)	✓
Does the compute model has scalability potential? (Scalability++)	
Does the compute model has potential for heterogeneous systems? (Coverage++)	✓
Is work published in rank A conference(s) or journal(s)? (Quality++)	

Application codes (AC) Specific Criteria

Does the compute model requires shared memory? (Scalability–)	
Does the compute model requires global communications? (Scalability–)	
Does the compute model exhibits SIMD parallelism? (Scalability++)	
Does the compute model exhibits threads parallelism? (Scalability++)	

Libraries and Frameworks (LF) Specific Criteria

How much of the expected functionalities are covered? (Coverage+= percentage)	0 100

Development Tools (DT) Specific Criteria

Consider Fortran? (Coverage++)	✓
Consider C/C++? (Coverage++)	✓
Consider MPI? (Coverage++)	✓
Consider debugging issues? (Coverage++)	
Consider performance monitoring/tracing? (Coverage++)	✓

Programming Model Implementation (PM) Specific Criteria

Is it part of a standardizing initiative? (Support++)	

Figure 2: Screendump of Demonstrator Website for an Exascale TRL Level 1 Assessment

The second part of the work agenda of the EESI2 project work package 6, called Task 6.2 ("Perform evaluation on 3 components"), scheduled for the second and third year of EESI2, is to

- Identify suitable EU Exascale projects, review their DOW and identify their important Exascale software components
  - Focus particularly on the first-phase EU FP7 Exascale projects
  - CRESTA, Mont-Blanc, DEEP, EPiGRAM, EXA2CT, NUMEXAS
- Select three software components
  - Ideally: components used and developed in more than one project

- Apply methodology defined in task 6.1 to selected components to determine and document their maturity level
- Document feedback on the methodology, best practices, suggestions for its improvement and further recommendations

4. Conclusions

This deliverable (D6.1) reports on the results of the process to develop and document a methodology for estimating the level of maturity of Exascale software components as defined by WP6 Task 6.1 (“Evaluation methodology set up”). This is a first step in investigating the feasibility and preparation for the operation of a European Exascale Software Centre which would coordinate research, development, testing, and validation of EU HPC Exascale software ecosystem components and modules.

We describe a draft proposal for a HPC software maturity level model. We define four main HPC software component classes, namely scientific application codes, libraries and frameworks, development tools, and programming model implementations, to clearly set the context for our proposal. Also, Exascale TRLs are proposed. The core of our model are the main quality factors which we believe are indicators of the maturity level of HPC software components and therefore can be used as assessment criteria for HPC software maturity. For each factor, we try to list directly measurable indicators or, if direct indicators do not exist, provide a list of “proxy” indicators which indicate the maturity. The main factors are documentation, support, availability, coverage, portability, scalability, performance, and quality.

Our model is based on experiences with various “Technology Readiness Level” (TRL) assessments used by various organisations in different technical fields (e.g., US DoD, US Air Force, NASA, European Space Agency) and the QualiPSo Open Maturity Model (OMM), one of the results of the EU FP6 project QualiPSo.