
*This document represents work in progress as the later additions
of a PRINCE2 process framework, extended assessment criteria,
and the support process are currently being refined*

OMII-UK Software Evaluation Process

Revision: 0.16

Authors:

Steve Crouch, Rob Baxter, Tim Parkinson

OMII-UK
32/4053
University of Southampton
Southampton
Hants., SO17 1BJ
Tel: +44 (0)23 80598862

Contents

<i>1</i>	<i>Introduction</i>	<i>3</i>
1.1	Purpose of Document.....	3
1.2	Motivations.....	3
<i>2</i>	<i>Overview: the OMII-UK Software Evaluation Process</i>	<i>3</i>
<i>3</i>	<i>Evaluation Process</i>	<i>4</i>
3.1	Tutorial-based Evaluation	7
3.1.1	Overview.....	7
3.1.2	Process	7
3.1.3	Outcomes	8
3.2	Criteria-based Assessment	9
3.2.1	Overview.....	9
3.2.2	Criteria	9
3.2.3	Process	14
3.2.4	Outcomes	14
<i>4</i>	<i>Support Process</i>	<i>14</i>
4.1	Determining the Level of Support	14
4.2	Classifying the Level of Support	15
<i>5</i>	<i>References</i>	<i>15</i>

1 Introduction

1.1 Purpose of Document

This document defines the process for evaluation of software delivered to OMII-UK, as well as the methods and criteria for evaluation.

The target audience for this document is developers wishing to provide software to OMII-UK for evaluation, and potential users wishing to understand the evaluation process. A separate document, the *OMII-UK Software Submission Guidelines* [1], provides best practice recommendations for how software should be packaged and provisioned for adopters, with OMII-UK able to provide assistance where required throughout the processes of packaging and evaluation to help improve the software.

1.2 Motivations

The primary motivation behind the OMII-UK evaluation process is to determine the level of maturity for selected software that is submitted to OMII-UK, and how it can be improved. Such software may be delivered as part of an OMII-UK Commissioned Software Project or from other sources.

This is achieved through two methods: tutorial-based evaluation and criteria-based assessment, detailed in Sections 4.1 and 4.2. Ideally, an evaluation period is 3-5 person-days, depending on the complexity of the software. The rationale is that if it takes longer than this to get the software up and running, there are clearly major issues that still need to be resolved. By assessing the software in a number of key areas, including usability at a number of technical levels, provisioning and support, these activities inform the decision to provide a level of support for the software.

2 Overview: the OMII-UK Software Evaluation Process

In general, a software release selected for evaluation by OMII-UK will go through the following process:

1. **Developers release version of software:** the software provider develops, tests, packages and releases a version of the software that ideally conforms with the *OMII-UK Software Submission Guidelines* [1].
2. **Evaluation:** with technical assistance from the developer team, OMII-UK evaluates the software to determine to what extent OMII-UK will support the software. This is an iterative process where any issues that are identified are passed back to the developer team, and responses evaluated, with the aim to complete a successful evaluation of the software. Section 3 covers the evaluation process and criteria in detail.
3. **Evaluation Report published:** the evaluation report is published on the OMII-UK website (see Section 4.1.3 *Outcomes*).
4. **Support:** informed by the software evaluation, OMII-UK provides a level of support for the software as an ongoing activity. Section 5 describes how support levels are determined, and what they mean.

If an evaluation is not successful, in that there were perhaps technical or legal issues that prevented a complete evaluation, then step 4 is omitted and a future release is awaited that addresses these issues.

After release, the developer team adopts a technical support role in the following phases:

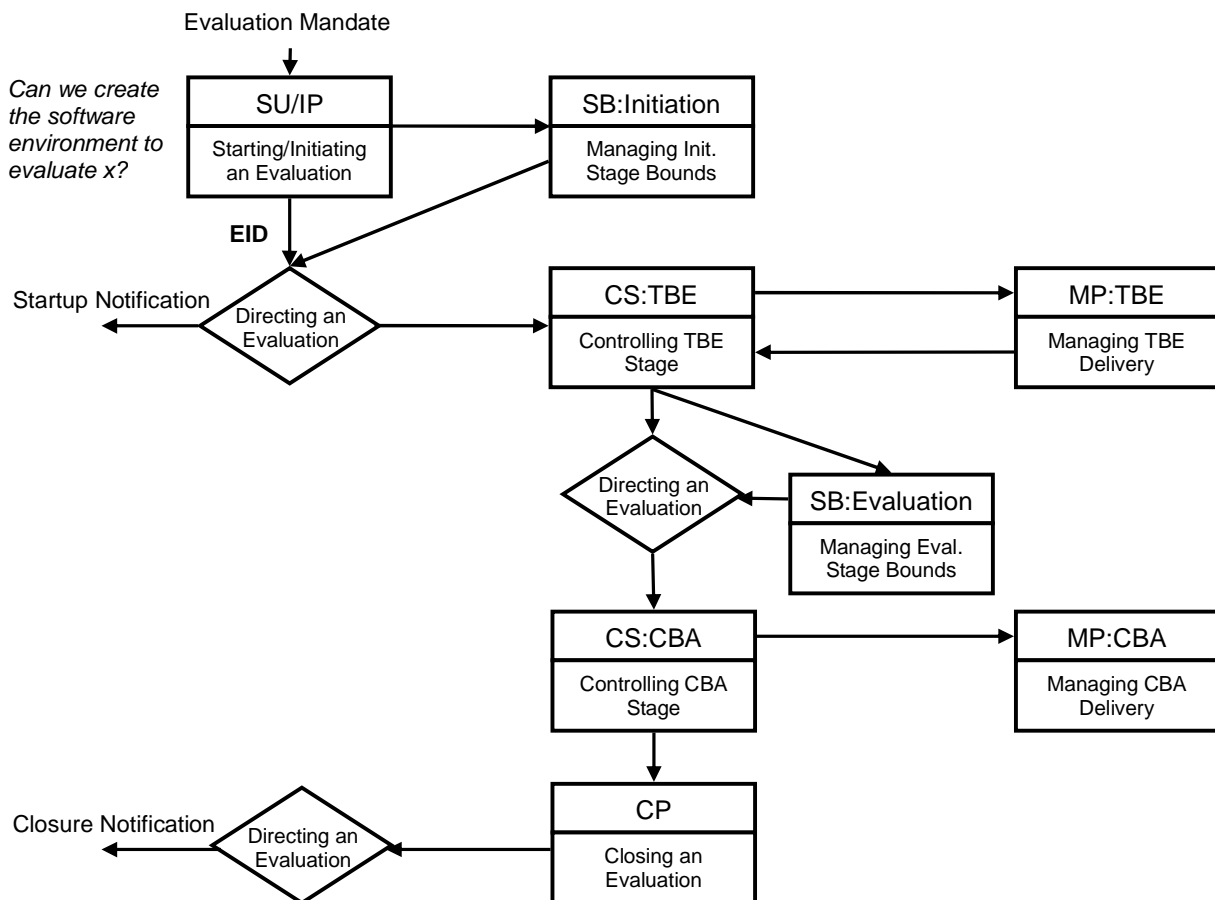
- **Evaluation support:** the developer team provides technical assistance during evaluation through their support policy and processes. A commitment to developer support, in terms of resources and timeframe, is agreed with the developer team prior to evaluation.
- **Post-release support:** assistance for user-level support after the release within the timeframe of the originating project.

Since an evaluation is a time-constrained activity, evaluation activities are agreed between management and evaluator as part of the start-up process, and time tolerances for each of these are defined. During an evaluation process activity, issues may be raised which require additional high-level guidance or to which there does not appear to be an adequate or appropriate resolution. The risk is that time tolerances for agreed evaluation activities may be exceeded. If such a risk is identified, the OMII-UK Operations Committee will decide on an appropriate course of action.

3 Evaluation Process

The evaluation process follows a tailored adoption of the PRINCE2 [2] project management method, with the aim to adopt those PRINCE2 processes, activities and deliverables that provide a suitable framework for managing evaluation activities.

An overview of the tailored PRINCE2 process is given in the diagram below, where the PRINCE2 processes are tailored to reflect management of evaluation. *The next sections offer a preliminary overview (to be extended and revised) for the major tasks in this process.*



3.1 Starting/Initiating an Evaluation:

This process is initiated following an evaluation mandate from the Board to the Evaluation Manager.

Due to the small scale that is required for managing evaluations, the 'Starting Up a Project' and 'Initiating a Project' processes have been combined into a single process. The goals of this activity are for the Evaluator (selected during this phase) to set up an appropriate environment for evaluating the software and to produce an Evaluation Initiation Document (EID).

Initially, the Evaluator attempts to set up an appropriate environment to evaluate the software, commissioning an appropriate test resource for installation, installing the software's pre-requisites and conducting an initial evaluation. If this is not possible, an Exception Report is generated to indicate problems with this process.

Following on from the setting up of a suitable evaluation environment, the EID is a short 2-page document produced by the Evaluator (based on a template) that describes:

- Based on the evaluation mandate, a short initial business case that focuses on evaluation motivations and benefits
- Evaluation approach: whether to perform both tutorial-based evaluation (mandatory) and criteria-based assessment (optional), and the particular aspects to focus on during evaluation
- Initial Project Plan for evaluation tasks including overall timeframe for evaluation and time tolerances for each evaluation task

- Initial evaluation risks
- Agreement with developers on developer personnel involvement (and other resources) necessary for conducting evaluation, with agreement on support availability timeframes
- Initial evaluation environment overview

3.2 Managing Initiation Stage Boundaries

This very short activity focuses on ensuring the EID is ready for the Board to make a decision on whether or not to proceed with the first stage of evaluation: *the tutorial-based evaluation*. Any additional direction for an evaluation from the Board is catered for within a revised Project Plan within the EID.

3.3 Controlling TBE Stage

Following authorisation of an evaluation, the Evaluation Manager liaises with the Evaluator throughout the process of tutorial-based evaluation. Where the evaluation is predicted to go outside of the time tolerances for the evaluation, this is escalated to the Board via an Exception Report detailing options for continuance or aborting the evaluation.

3.4 Managing TBE Delivery

The Evaluator conducts the tutorial-based evaluation. See section 4.1 for an overview of this evaluation methodology. The Evaluator produces a Tutorial-based Evaluation Report based on this activity.

3.5 Managing Evaluation Stage Boundaries

This very short activity focuses on ensuring the Tutorial-based Evaluation Report is ready for the Board to make a decision on whether or not to proceed with the next stage of evaluation: *criteria-based assessment*. Any additional direction for an evaluation from the Board is catered for within a revised Project Plan.

3.6 Controlling CBA Stage

Following authorisation of an evaluation, the Evaluation Manager liaises with the Evaluator throughout the process of criteria-based assessment. Where the evaluation is predicted to go outside of the time tolerances for the evaluation, this is escalated to the Board via an Exception Report detailing options for continuance or aborting the evaluation.

3.7 Managing CBA Delivery

The Evaluator conducts the criteria-based assessment. See section 4.2 for an overview of this evaluation methodology. The Evaluator produces a Criteria-based Assessment Report based on this activity.

3.8 Closing an Evaluation

This stage is concerned with decommissioning evaluation test resources and in the event of natural evaluation closure, publicising the Tutorial-based Evaluation and Criteria-based Assessment reports.

4 Evaluation Methodology

The evaluation methodology is comprised of two core activities: tutorial-based evaluation and criteria-based assessment. The outcome of the evaluation process is a two-part report that covers these aspects, covered in the next two subsections, that informs the OMII-UK decision to support the software.

4.1 *Tutorial-based Evaluation*

4.1.1 Overview

This provides a pragmatic evaluation of the software intended for those assessing its practicality and technical suitability for use. It explains which aspects of the software functioned as expected and which do not, from the point of view of the evaluator.

Tutorial-based evaluation is a reflection of the process the evaluator undertook to learn, build, install, configure and use the software. It results in a pragmatic and candid report based on experiences, explaining what was achieved with the software following its documentation. It also includes the issues and shortcomings as observed, and can also include suggested workarounds and fixes for identified issues where they exist.

In addition, the tutorial-based evaluation can function as a practical guide for getting the software to work based on the experiences of the evaluator.

4.1.2 Process

The first part of this process is largely dependent on following the supplied documentation in a logical (or suggested) fashion to achieve eventual successful use of the software. 'Successful use' is split into two categories:

- **Basic:** software used to perform basic function(s) of the software requiring a minimum of preparation and technical expertise, e.g. can be performed rapidly following installation and configuration of required components via command-line or GUI-based tools.
- **Advanced:** software used to perform more advanced function(s) of the software, perhaps requiring a larger degree of preparation and technical expertise e.g. following install, further configuration of the software and required components is required, as well as the writing, compilation and execution of a Java example that uses the software via a supplied API.

In each case, successful use is therefore defined as being able to execute an example task (or tasks) of the software following given instructions, and successfully validate the results against expected output. Such example tasks should be representative of the core functionality of the software and its intended purpose.

Also investigated is to what extent the software is supported for users and developers through an open development process. To what extent is the following provided:

- **Source code repository:** are developers able to access either an up-to-date CVS or Subversion (SVN) code repository for source code? Does a policy exist for committing updates to the code base for those wishing to contribute to improving the software?
- **Packaged, usable software:** are the binary releases packaged for immediate use in a suitable archive format, e.g. .tar.gz for Linux, .zip for Windows?

- **Overview web page:** is download access for the software provided on a web page that includes an introduction to the software and how to get started?
- **Public bug/issue tracker:** are users/developers able to view existing software issues and add their own to a suitable tracking system?
- **Mailing lists:** do software announce and developer mailing lists exist for those wishing to be kept informed as to releases and developments of the software, or wishing to join discussions?

Importantly, to what extent are these capabilities provided for in the future? i.e. beyond the lifetime of any project from which the software originates?

During this process, there are close ties between the evaluator and the developer team. As issues are discovered, they are reported to the developer team and discussed. Ideally, the issues are entered directly into the developer's bug tracking system (e.g. SourceForge issue tracker, Bugzilla, JIRA) if one exists for traceability. Once received, the developer team is then able to address the issues in a future release, and supplying workarounds where they exist in the meantime to ameliorate the problems.

4.1.3 Outcomes

The outcomes from this process are two-fold. Firstly, encountered issues reported directly to developer team as they occur (ideally using the developer team's issue reporting system if one exists). Secondly, an evaluation report is produced that details the process undertaken to use the software, and the experiences of that process (which includes the discovered issues). For example, a typical initial report may be structured thus:

- 1 Introduction [very brief overview of the software and purpose]
- 2 Building x
 - 2.1 Build Pre-Requisites [perhaps issues of obtaining some software]
 - 2.2 Building the Software [e.g. not appropriately documented]
 - 2.3 The a Building Problem [fixable through discussion with developers]
 - 2.4 How to Address a Building Problem [the supplied workaround]
- 3 Installing x
 - 3.1 Install Pre-Requisites [well explained]
 - 3.2 Installing the Software [straightforward]
 - 3.3 Configuring the Software [needs more detail in some parts]
- 4 Performing y with x [basic, achieved]
- 5 Performing z with x [advanced, non-achieved]
 - 5.1 The b Problem with Performing z with x
- 6 Product Accessibility and Support for an Open Development Process
 - 6.1 Source Code Repository
 - 6.2 Packaging of Software
 - 6.3 Introduction/Overview 'first look' web page
 - 6.4 Public Bug/Issue Tracker
 - 6.5 User/Developer Mailing Lists
- 7 Discovered Issues [discovered issues are iterated here and referenced in document]

Because of the style of the report, appropriate content can be fed back into the software documentation directly by the developers to improve it.

A subsequent evaluation report for a later release of the software would be a separate, updated version of the previous report, including the issues previously encountered and to what extent they are resolved.

4.2 Criteria-based Assessment

4.2.1 Overview

This is a quantitative assessment of the software intended for those wanting to rapidly assess properties of the software in terms of sustainability and maintenance, functional characteristics and usability. It gives a measurement of quality in a number of areas, which can inform a high-level decision on whether to adopt the software.

The maturity of the software against its intended function is a key consideration when determining its quality. A mature, stable, quality product that fulfils only a subset of community requirements will score higher than a more ambitious product of prototype quality that attempts to meet a much larger set of community requirements.

4.2.2 Criteria

The criteria for assessment (partially based on ISO 9126), are as follows:

Criteria	Sub-criteria	Notes – to what extent is/does the software...
Usability	Understandability	Easily understood?
	Buildability	Straightforward to build on a supported system?
	Installability	Straightforward to install on a supported system?
	Learnability	Easy to learn how to use its functions?
	Operability	Easy to use?
	Timeliness	Complete tasks in a timely manner?
Functionality Characteristics	Accuracy	Produce expected results?
	Interoperability	Interoperate with other required software?
Sustainability & Maintainability	Community	Have evidence of current/future community?
	Licensing	Adoption of appropriate licence?
	Accessibility	Have evidence of current/future ability to download?
	Testability	Easy to test correctness of source code?
	Portability	Usable on multiple platforms?
	Supportability	Have evidence of current/future developer support?
	Analysability	Easy to understand at the source level?
	Changeability	Easy to modify & contribute changes to developers?
	Evolvability	Have evidence of current/future development?

These criteria, and their sub-criteria, are currently work in progress, and require refinement, with input and feedback from software developer and user communities.

4.2.2.1 Usability

Understandability: how straightforward is it to understand:

- What the software does and its purpose?
- The intended market and users of the software?
- The software's basic functions?
- The software's advanced functions?

Buildability: how straightforward is it to:

- Meet the pre-requisites for building the software on a build platform?
- Build the software on a build platform?

Installability: how straightforward is it to:

- Meet the pre-requisites for the software on a target platform?
- Install the software onto a target platform?
- Configure the software following installation for use?
- Verify the installation for use?

Learnability: how straightforward is it to learn how to achieve:

- Basic functional tasks?
- Advanced functional tasks?

Operability: how straightforward is it to:

- Perform basic functional tasks?
- Perform advanced functional tasks?

Timeliness: to what extent does the software operate in a timely manner?

Sub-criteria	Assessment Mark Guidelines (0-10):
Understandability	0 – no adequate descriptions of software
	3 – some vague descriptions of basic functional use
	5 – adequate descriptions for purpose and basic functional use; architectural overview; appropriate diagrams
	7 – excellent descriptions for intended software market/domain and purpose, basic/advanced functional use; clear architectural overview and third party software interactions, some use cases explained
	10 – project roadmap, good descriptions of use cases based on real deployments
Buildability	0 – cannot meet build pre-requisites – third party software unavailable
	3 – build pre-requisites difficult to meet and/or building unnecessarily difficult
	5 – building straightforward
	7 – building easy and largely automated
	10 – building fully automated
Installability	0 – cannot meet install pre-requisites – third party software unavailable
	3 – install pre-requisites difficult to meet and/or installation and configuration unnecessarily difficult
	5 – installation straightforward but configuration process difficult
	7 – installation and configuration largely automated, verification process for installation provided
	10 – clear, easy, fully automated install and verification process
Learnability	0 – instructions for basic use completely inadequate or absent

	3 – some instructions for basic use of the software
	5 – concise, clear instructions for basic use with examples, some instructions for advanced use
	7 – getting started guide and some tutorials for basic use, clear instructions for advanced use; clear examples for basic and advanced use with expected output
	10 – excellent basic and advanced tutorials covering common use cases of the software with appropriate supporting material
Operability	0 – impossible to perform basic tasks
	3 – can perform basic tasks (e.g. command line use), but process unnecessarily complex
	5 – straightforward to perform basic tasks with given examples
	7 – can perform advanced tasks (e.g. use of supplied API), but process unnecessarily complex
	10 – very straightforward to perform advanced tasks with given examples
Timeliness	0 – operation consumes inordinate amount of time for all functions
	3 – operation consumes ordinate amount of time for some basic functions
	5 – operation consumes ordinate amount of time for most basic functions
	7 – operation consumes ordinate amount of time for most basic and some advanced functions
	10 – operation consumes ordinate amount of time for most basic and advanced functions

4.2.2.2 Functionality Characteristics

Accuracy: can the software be verified to produce expected results?

Interoperability: to what extent does the software's interoperability:

- Meet appropriate open standards?
- Function with required third-party components?
- Function with optional third-party components?

Sub-criteria	Assessment Mark Guidelines (0-10):
Accuracy	0 – software cannot be verified to produce expected results, or expected results not supplied
	3 – software does not meet expected results for basic tasks
	5 – software meets expected results for basic tasks, no expected results for advanced tasks
	7 – software meets expected results for advanced tasks
	10 – accuracy verification suite provided for basic and advanced tasks
Interoperability	0 – all proprietary and bespoke
	3 – some open standards adopted
	5 – appropriate open standards adopted, but draft, extended or non-ratified
	7 – primarily based on appropriate, mature open standards, evidence of demonstrable interoperability against other compliant software
	10 – demonstrable interoperability against other compliant software, completely based on appropriate, mature open standards

4.2.2.3 Sustainability and Maintainability

Community: to what extent does/will an active user community exist for this product?

Accessibility: to what extent is the software accessible?

Supportability: to what extent will the product be supported currently and in the future?

Testability: how straightforward is it to test the software to verify modifications?

Portability: to what extent can the software be used on other platforms?

Analysability: how straightforward is it to analyse the software's source release to:

- To understand its implementation architecture?
- To understand individual source code files and how they fit into the implementation architecture?

Changeability: how straightforward is it to modify the software to:

- Address issues?
- Modify functionality?
- Add new functionality?

Evolvability: to what extent will the product be developed in the future:

- For a future release?
- Within a roadmap for the product?

Sub-criteria	Assessment Mark Guidelines (0-10):
Community	0 – no discernible user community
	3 – evidence of small user community; e.g. 1-2 research group-scale use
	5 – established user community within a single discipline; e.g. multiple institutional-scale use
	7 – active national user community across multiple disciplines; e.g. national-scale use (e.g. NGS)
	10 – thriving national (or global) user community across many multiple disciplines; e.g. international-scale use (e.g. NGS, EGEE, TeraGrid, etc.)
Licensing	0 – proprietary or commercial licence
	3 – open licence but for binaries only, licence not recognised by Open Software Foundation (OSF) or limited in openness (e.g. LGPL)
	5 – open but viral (e.g. GPL), licence recognised by OSF
	7 – open, non-viral licence
	10 – fully permissive open licence (e.g. BSD)
Accessibility	0 – source and binaries not freely and publically accessible, accessibility completely unsustainable (e.g. only by prior email)
	3 – binaries and/or source accessible from an unsustainable resource (e.g. user's web site); perhaps overly complex pre-download registration process; evidence for > 6 months accessibility
	5 – freely and publically accessible source and binaries from a single project website, but no evidence of sustainability beyond project lifetime; adoption of a source version control system (e.g. SVN, Mercurial, Git, CVS); any pre-download registration process straightforward; evidence for > 1 year accessibility
	7 – accessible from multiple locations, academic/institutional resource with

	evidence of accessibility beyond contributing project's lifetime; evidence for > 2 years accessibility
	10 – unrestricted download from a recognised site with a sustainable future, multiple mirrors of software portfolio (e.g. SourceForge), feasible access sustainability plan; evidence for > 5 years accessibility
Supportability	0 – no evidence of any current or future developer support
	3 – single developer support contact provided (e.g. via single email address), no evidence of developer support beyond project lifetime
	5 – multiple developer support provided (e.g. mailing list), issue tracking system with evidence of use, evidence of developer support beyond project lifetime
	7 – scalable developer support provided (e.g. support ticketing system) with dedicated support team with strong evidence of a future beyond project lifetime, commitment to move towards open development model
	10 – adoption of comprehensive, feasible and sustainable open development model with evidence of ability to deliver; active open source mailing list, several stakeholders with vested interest, broad range of committers
Testability	0 – no tests exist
	3 – some manual tests exist to verify basic functionality
	5 – some automated tests exist for basic functionality (e.g. JUnit tests)
	7 – full automated test suite for basic functionality
	10 – test suite includes code coverage analysis
Portability	0 – only operable on a single distribution/variant of a particular operating system (e.g. Linux Debian, Windows XP); if browser-based operation, only operable on single type of recognised browser (e.g. Firefox, Chrome, Internet Explorer, Opera)
	3 – operable on good selection of recent, available multiple distributions/variants of a particular operating system, but functional restrictions apply on some variants (e.g. Linux Debian, Ubuntu, RedHat, SuSE or Windows XP, Vista); if browser-based operation, operable across 1-2 recognised browsers
	5 – fully operable on multiple distributions/variants of a particular operating system; if browser-based operation, operable across most recognised browsers
	7 – operable on major recognised operating systems (e.g. Linux, Windows, Mac OSX), but functional restrictions apply on some operating systems or operating system variants; if browser-based operation, operable across vast majority of recognised browsers
	10 – fully operable on major recognised operating systems/browsers
Analysability	0 – source code unavailable or source architecture completely unclear
	3 – source architecture mainly clear, inappropriate/monolithic modularity, poor code quality and sporadic source commenting, difficult to map basic software functionality to source, perhaps deprecated/inactive code branches; JavaDoc with limited package and class descriptions
	5 – source architecture clear, reasonable modularity and code quality, adequate source commenting, can map basic and advanced functionality to source, but not importable to recognised Integrated Development Environments (IDEs) (e.g. Netbeans, Eclipse), perhaps some evidence of inactive/deprecated source branches; JavaDoc with thorough package and class descriptions
	7 – excellent modularity and code quality, extensive source commenting, importable to a recognised IDE, no inactive/deprecated source branches
	10 – full IDE project support
Changeability	0 – cannot contribute source code modifications to software project
	3 – architecture not easily extensible; no documented process to contribute code changes
	5 – appropriately extensible architecture for incorporating new functionality, basic

	documented process to contribute code changes to main code base
	7 – documented code contribution/feedback policy and process for submitting changes to software's source versioning control system
	10 – full Open Development model
Evolvability	0 – no evidence of future development
	3 – some evidence of future development planning, and history of software releases addressing issues, but no evidence of sustainable development post-project; closed in-house development
	5 – strong evidence of future development plans to address user community requirements and issues, with track record of previous software releases addressing previous community requirements and issues; open 'lazy consensus'/'benevolent dictator' approach to product development
	7 – open, regularly publicised, community-focused product roadmap; strong, active, fully open consensus of product development
	10 – self-regulatory, self-sustaining exemplar of Open Development

4.2.3 Process

This assessment is conducted in parallel with the tutorial-based evaluation, which informs this assessment.

4.2.4 Outcomes

The outcome from this process is a completed assessment form [to be created!].

5 Support Process

5.1 Determining the Level of Support

In order to determine the level of support to provide, firstly the operational quality and usability of the software is evaluated. The intent is to answer the following questions:

- To what extent can the software be understood through its documentation?
- To what extent are the software pre-requisites and processes for installing and configuring them clearly and concisely explained?
- To what extent is the software straightforward to build, install and configure for use by an appropriately skilled individual?
- To what extent is the software straightforward to use?

From a technical support perspective:

- What technical or other aspects of the software need to be addressed in the short term, medium term and longer term that represent barriers to the software's maturation in the future?
- How straightforward is it to modify the software to address issues or meet new requirements?

During this evaluation process, identified issues are passed back to the developer team for resolution.

In addition, the following issues are considered in terms of provision for support:

- To what extent is the software supported by an open development process? e.g. the existence and support for a publically accessible source code repository with a code contribution policy, issue tracking system and mailing lists, etc.
- What level of future development and support is expected from the developers of the software? How will the level of support change in the future?

User demand for the software from established research communities will also be taken into account when deciding on the level of support to provide i.e. the software already has a large established user base, or there is sufficient user demand for initial uptake.

5.2 Classifying the Level of Support

Based on an evaluation of a piece of software, the OMII-UK Operations Committee decides on an appropriate level of support to be provided by OMII-UK in a number of areas:

Support Class		Level of OMII-UK Effort For ...		
		Development	Support	Promotion
★★★★★	Developed	Funded	Funded	Funded
★★★★☆	Supported	Unfunded	Funded	Funded
★★★☆☆	Sustained	Unfunded	Unfunded	Funded
★★☆☆☆	Mothballed	Unfunded	Unfunded	Unfunded
★☆☆☆☆	Deleted	Unfunded	Unfunded	-Funded

It is anticipated that the level of provided support for a piece of software will change as the software evolves over time and is subjected to further evaluations.

This will be refined with input and feedback from software developer and user communities, including descriptions of transitions between each of these support classes as software, and required support, evolves and matures.

6 References

- [1] *OMII-UK Software Submission Guidelines*, S. Crouch et al, OMII-UK, August 2009
- [2] *Managing Successful Projects with PRINCE2 (PROjects IN Controlled Environments)*, TSO (The Stationary Office), ISBN 9780113309467, 2007.