Fundamental SKA Software Standards

Glossary

In this document, terms in bold italic font are terms that have a specific meaning in the document. The first usage of the term in anything will generally represent a definition. For example, in this document:

- ***Software*** includes all firmware as well as traditional software, in both source code and binary code form.
- ***SKA Software*** is ***software*** that is essential for the SKA Observatory to be supported and operated.

Introduction

Purpose of the document

This document outlines standard that are applicable to all SKA software.

Scope of the document

The scope of these standards includes all SKA Software and the infrastructure associated with it.

SKA software lies on a spectrum comprising:

1. ***Off-the-shelf software*** is software that was not written specifically for the SKA. This includes, for example:
    1. Operating systems
    2. Compilers
    3. Database software
    4. Desktop applications
2. ***Derived software*** is software that has some modules written or modified explicitly for SKA but which also includes some modules that were originally developed for some other purpose. Examples include:
    1. Framework software such as the Tango control system.
    2. Business software such as procurement software which may be heavily customised for the SKA Organisation.
3. ***Bespoke software*** is software that has been written specifically for the SKA. This includes, for example:
    1. Control and monitoring software such as Tango device servers.
    2. Data-driven data processing software for SDP and the Non-Imaging processing software of CSP.
    3. Web based software with database backends for observation and user management.
    4. FPGA Firmware for LFAA and CSP.

Derived software is a continuum that ranges all the way between pure ***off-the-shelf software*** and pure ***bespoke software***. and the standards that are applicable are also a mixture of the standards for the two extremes.

In the following there are explicit standards that apply to ***off-the-shelf software*** and ***bespoke software***. In general, the former will apply to the non-SKA modules of derived software, and the latter to SKA modules. However, the area is complicated (for example, some open-source off-the-shelf components may satisfy most of the bespoke software requirements) and so these standards must be applied intelligently as guidelines. In most cases a common-sense approach can be taken.

References

Applicable documents

The following documents are applicable to the extent stated herein. In the event of conflict between the contents of the applicable documents and this document, **the applicable documents** shall take precedence.

- [AD1] SKA.GOV.POL-SKO-POL-001 SKA Intellectual Property Policy. Available from: https://www.skatelescope.org/wp-content/uploads/2011/03/SKA-GOV.POL-SKO-POL-001_IPpolicyRevX.pdf

Reference documents

The following documents are referenced in this document. In the event of conflict between the contents of the referenced documents and this document, **this document** shall take precedence.

- [RD1] ISO/IEC 12207:2013 Systems and software engineering – Software life cycle processes
- [RD2] ISO/IEC/IEEE 15288:2015 Systems and software engineering -- System life cycle processes
- [RD3] Clement et al, Documenting Software Architectures: Views and Beyond, Second Edition Addison-Wesley, 2011
- [RD4] ISO/IEC/IEEE 42010:2011(E), Systems and software engineering — Architecture description 2012
- [RD5] RFC 5424 - The Syslog Protocol, IETF Network Working Group
- [RD6] IEC 62682:2014 Management of alarms systems for the process industries

Standards applicable to all SKA software

1. All ***SKA software*** shall have a copyright notice which is a description of who asserts the copyright over the software.
    a. Notes:
        i. ***Derived software*** and ***bespoke software*** will normally have mixed copyright.
2. All ***SKA software*** shall have a ***software license*** which is a legal instrument governing the use or redistribution of software.
    a. Notes:
        i. Off-the-shelf software will normally have licenses over which the SKA has no control.
        ii. Derived software may have mixture of licenses.
        iii. Bespoke software will normally have a permissive open source license.
3. The documentation associated with ***SKA software*** should also carry a license unless it is covered by the software license.
4. All ***software licenses*** governing a body of software must be mutually compatible.
5. All ***software licenses*** for ***SKA software*** should be agreed with the SKA Organisation prior to the software being adopted or developed.
    a. Notes:
        i. The SKA Organisation will always agree to a 3 clause BSD license for software (provided there are no compatibility issues) and will favour open-source permissive licenses with attribution since they minimise compatibility issues.
        ii. The SKA Organisation will always agree to a Creative Commons Attribution 4.0 International License for documentation.
        iii. This permissive open source recommendation is significantly more permissive than the SKA IP policy [RD1] which only requires contributors to "grant non-exclusive, worldwide, royalty-free, perpetual, and irrevocable sub-licences to other SKA Contributors to use those innovations and work products for SKA Project purposes only."

Standards applicable to Off-the-shelf software

All ***off-the-shelf software*** should have:

1. All ***off-the-shelf software*** should have a business case describing the requirements for the software and the other software considered.
2. All ***off-the-shelf software*** should have evidence that the software meets these requirements.
3. All ***off-the-shelf software*** should have a description of how the software will be supported during the expected lifetime of the software,
    a. Notes:
        i. The SKA Observatory has a predicted lifetime of 50 years, which is much longer than most software products and the companies that develop most of them. Hence this description may include how many alternatives exist which also

support the software's data products, escrow agreements, commercial soundness of the company. Support includes:

1. Managing unexpected behaviour of the software that is incompatible with SKA's (possibly evolving) requirements.
2. Managing the evolution of underlying systems, such as hardware and operating systems, that the software relies on.
3. Managing changes to the existing supplier support arrangements (e.g. original company being bought our, product becoming not commercially viable etc).

4. All ***off-the-shelf software*** should have evidence that the software has been developed to a standard of quality appropriate to the needs of the SKA Organisation.
5. All ***off-the-shelf software*** should have documentation that is appropriate to the needs of the SKA Organisation.
6. All ***off-the-shelf software*** should have been approved by the SKA Organisation as to its fitness for purpose and included in a public register of approved SKA Software.

Standards applicable to bespoke software

Design

This section comprises standards relating to processes described by RD1, ISO 12207 (2008) §7.1.2 (Requirements), §7.1.3 (Architecture) and §7.1.4 (Detailed Design). They complement any general System Engineering level standards (i.e. processes relating to ISO 15288 [RD2]) applicable to all SKA systems.

All bespoke software should have documentation, models and prototypes covering the following:

1. The requirements the software is intended to fulfil.
2. The software architecture used.
    a. Notes:
        i. The software architecture is the primary deliverable at CDR. Detailed design is only required to the extent needed for reliable cost estimation.
        ii. The recommended reference for architecture documentation is "Documenting Software Architectures: Views and Beyond, Second Edition" (Clements et al, 2011) [RD3]. This book should be consulted for best practices on documenting views, styles and interfaces. The ISO 42010 [RD4] standard is also relevant.
        iii. The architecture documentation should include, at minimum
            1. System Overview, including a description of the architectural styles used.
            2. A set of views describing key features of the architecture, and the mapping between views.
            3. Interface Documentation or references to applicable Interface Control Documents for the major interfaces.

4. Rationale justifying how the architecture meets the requirements. Justification on the basis of models and evolutionary prototypes is highly recommended in many cases.
5. A consideration as to whether there is any existing software that meets, or can be modified to meet, the requirements.

iv. Emphasis should be on clear, unambiguous diagrams with accompanying descriptions and tables.

v. Refer to Chapter 11 of Clements et al for a description of interface documentation. Interfaces that are language or framework specific may be best documented in a format appropriate to that language or framework (e.g. generated from comments and code in an evolutionary prototype).

3. Detailed design of components.
    a. Note:
        i. It is expected and encouraged that most of the detailed design may be automatically generated from code and comments.

The software design should be reviewed and the reviews should incorporate the following factors:

4. SKA Organisation is responsible for L1 requirements and must agree and review all L2 and L3 requirements.
5. SKA Organisation personnel should be involved in software architecture reviews
6. Detailed design should be reviewed:
    a. By someone in addition to the principal developer of the module being considered.
    b. In a manner appropriate to the significance of the module.
        i. Note:
            1. The significance of the code relates to the impact any changes to the design has on other parts of the system.
            2. The review process must not be overly bureaucratic. Development teams should be empowered to design and develop the code efficiently and modify the internal design when required.

Construction

This section comprises standards relating to processes described by ISO 12207 (2008) §7.1.5 (Construction).

The construction of all ***bespoke software*** should include:

1. The construction of all source code should follow a defined documented process that is approved by the SKA Organisation.
2. All construction should utilise an SKA Organisation approved version control system.
    a. Note:
        i. The SKA Organisation approved version control system is Git.

3. All documentation source code, software source code, firmware source code, unit tests, build scripts, deployment scripts, testing utilities and debugging utilities must reside in the version control repository.
4. Software should be written in an SKA approved language and adhere to SKA language specific style guides.
    a. Note:
        i. The primary approved language will be Python.
        ii. The coding standards for Python will be adapted from the LSST DM code style guides.
        iii. Use of other languages will have to be justified by, for example:
            1. It is not possible to run Python in the chosen run-time environment.
            2. Python doesn't provide the necessary performance.
        iv. Many other languages are likely to have extensive usage. For example:
            1. C++ (for high performance computation on conventional CPU's).
            2. Java (e.g. for business logic in web systems and derived software).
            3. VHDL (for FPGA firmware).
            4. CUDA (for GPU software).
            5. Javascript (for Web client systems).
5. SKA Organisation employees must have access to the repository while the software is under development, be able to sign-up for notifications of commits and if necessary give feedback to the developers.
6. Source code should include unit tests at the class, function or source file level to test basic functionality of methods (functions) with an agreed minimal coverage (between 75 and 90%). Unit tests created for fixing defects or making specific enhancements should be checked-in with a reference to the issue for which the tests were created.
7. A workflow that provides the following support for work management:
    a. All work tasks should be described in a ticketing system.
    b. Work tickets should have a description of the task, an estimate of the resource required and amount of the task that has been completed.
    c. All code commits should relate to a ticket in the ticketing system.
    d. The developing organisation should be able to use the ticketing system to generate progress metrics.
8. A workflow that ensures the following support for code on the main development branch of the development party:
    a. With the exception of trivial cases (e.g. possibly documentation changes) code can only be added to or merged with the main development branch by a pull-request or similar mechanism.
    b. The pull request (or similar mechanism) must only be accepted after the code has been cleanly compiled and passes all appropriate tests. This process should be triggered automatically.
    c. Pull requests must only be accepted after the code changes has been reviewed by more than one developer (inclusive of the primary developer).
    d. Pull requests must only accepted by suitably qualified individuals.

9. Test software verifying the system software at multiple levels (from the complete system down to individual module unit tests). These tests should often be traceable to specific requirements and, as far as practicable, be able to be run automatically.
10. Software simulations/stubs/drivers/mocks for all major interfaces to enable sub-system and system level tests.
11. Automated documentation generation - including, but not limited to, of detailed design documentation.
12. A complete definition of other software (both off-the-shelf and bespoke) that the software requires to build and deploy.
13. Deployment scripts or configurations, which allows the software to be deployed cleanly starting with a bare deployment environment.
14. The ability to log diagnostic information using a RFC 5424 “Syslog” protocol.
15. The ability, dynamically at runtime, to suppress or select logging of messages at different Syslog severity levels on at least a per-process basis (and a per-thread basis or per class basis if appropriate).
16. The ability to log diagnostics at all major interfaces at a RFC 5424 Debug severity level.
17. Alarms, where applicable, should be based on the IEC 62682 standard [RD6].

Acceptance and handover

This section comprises standards relating to processes described by ISO 12207 (2008) §6.4.8 (Acceptance Support), §7.1.6 (Integration) and §7.1.7 (Qualification).

Bespoke software will only be accepted by the SKA Organisation after it has been appropriately integrated and validated. The integration and validation of ***bespoke software*** must include:

1. The integration, validation and acceptance of all source code should follow a defined documented process that is approved by the SKA Organisation.
2. This process must make clear, for all times during the handover:
    a. Who is responsible for making software changes.
    b. What the expected turnaround time for software changes is.
3. Code has been shown to pass appropriate, system, sub-system and unit level tests.
4. Code has been checked into an approved SKA acceptance repository.
5. Software will be integrated, as far as possible, prior to SKA Array Acceptance schedule. This will be done by a series of software integration “Challenges” which predate IET, and continue through the array release period.
6. When the SKA Organisation takes over maintenance of the software the complete repository including commit history must be delivered to the SKA Organisation.
7. Where code requires specialist hardware for testing, that hardware will be included as part of the handover.

Support Infrastructure

To develop and integrate software SKA will provide:

1. A central, globally visible, set of repositories that can be used by any SKA developers.
2. A globally accessible website for the storage and access of documentation.
3. A continuous integration and test framework that is open to use by developers. This will include at support for at least the 4 types of ***bespoke software*** described in the scope section (Tango, SDP and NIP data driven software, Firmware and Web Applications).
4. Communication tools to enable software developers to access expertise from all the SKA software developer community.

Appendix 1. Preliminary list of approved off-the-shelf software

Note.

- This section is a WIP so is not meant to be reviewed yet.
- This is not meant to be a restrictive list at the moment - it is just to get things started. The most important thing at the moment is to identify the software that is in use at the moment and identify any controversial areas and reduce duplication of functionality

Bespoke software tools

1. Debian and Ubuntu operating systems and their associated packages.
2. The Tango control system framework.
3. Jira
4. Git
5. Read-the-docs/Sphinx
6. Python and associated packages and development tools
7. C/C++ tools

Documentation and communication tools

1. Confluence
2. MS Office suite of software.