SKA SOFTWARE ENGINEERING PROCESS HARMONISATION

Document number SKA-TEL-SKO-0000558
Context ..... PPP-PPP-PPP-TTT
Revision 1.0
Author ..... Nick Rees
Date 2015-06-06
Document Classification ..... UNRESTRICTED
Status ..... Draft

Name	Designation	Affiliation	Signature	
Authored by:				
Nick Rees	Head of Computing and Software	SKAO		
			Date:	2016-06-03
Owned by:				
			Date:	
Approved by:				
			Date:	
Released by:				
			Date:	

DOCUMENT HISTORY

Revision	Date Of Issue	Engineering Change Number	Comments
0.1	2016-05-25	-	First draft release for internal review
0.2	2016-06-01		After suggestions from SE and presentation to EIT meeting
1.0	2016-06-03		Reformatted to SKA template.

DOCUMENT SOFTWARE

	Package	Version	Filename
Wordprocessor	MsWord	Word for Mac 2015	SKA-TEL-SKO-0000558-Rev1 SKA Software Engineering Process.docx
Block diagrams			
Other			

ORGANISATION DETAILS

Name	SKA Organisation
Registered Address	Jodrell Bank Observatory Lower Withington Macclesfield Cheshire SK11 9DL United Kingdom Registered in England & Wales Company Number: 07881918
Fax.	+44 (0)161 306 9600
Website	www.skatelescope.org

TABLE OF CONTENTS

LIST OF FIGURES

No table of figures entries found.

This is an automatic table of contents. To use it, apply heading styles (on the Home tab) to the text that goes in your table of contents, and then update this table.

If you want to type your own entries, use a manual table of contents (in the same menu as the automatic one).

LIST OF TABLES

No table of figures entries found.

This is an automatic table of contents. To use it, apply heading styles (on the Home tab) to the text that goes in your table of contents, and then update this table.

If you want to type your own entries, use a manual table of contents (in the same menu as the automatic one).

LIST OF ABBREVIATIONS

AN..... Another

EX..... Example

SKA ..... Square Kilometre Array

SKAO ..... SKA Project Office

1 Introduction

1.1 Purpose of the document

Part of the role of the SKAO as the SKA design authority is to understand the diversity of the software and computing systems that are being proposed and attempt to harmonize these systems across the elements in order to reduce the diversity of processes and systems that will be delivered to the AIV and operations teams. A key part of this is to harmonize the software engineering processes (as much as is practicable) so that when software is delivered it is delivered in a consistent way, to an agreed set of standards. Many of computing elements of the project have indicated a desire to move towards a more agile form of development, but no processes have, as yet, been defined as to describe what this means. This document outlines a development to define a basic set of requirements for all software deliveries, and define processes for software engineering in the SKA context.

1.2 Scope of the document

The scope of the SKA Software Engineering process could vary considerably, partly because there is a grey area dividing software, firmware and hardware. It is relatively easy to define many aspects of the process if the scope is kept small (i.e. only for standard software and computer architectures), but gets more difficult at scale and for non-standard architectures (GPU and FPGA development being different from CPU). Even so, it is still important that we understand the engineering process for these more difficult cases, and understand the development model, particularly if a non-waterfall approach is being proposed. Hence, the end result may be a layered process. More esoteric developments have to justify and outline their dependencies and how the life cycle of their system will be managed. Other developments, which use a more standard toolset, will have to share a development environment, processes and dependencies that are harmonized across many elements.

There is also overlap with the Consortia's CDR Statement of Work for the CDR deliverables, which includes the following three deliverables (based on guidance from the ECSS standards, ECSS-E-40 Part 2 in particular):

- Software architecture models and use cases
- Software libraries required to demonstrate compliance
- Software configuration information (operating system, libraries, compilers etc.) for demonstration of compliance.

This proposal builds on these deliverables, but is more encompassing and also strives for horizontal harmonization across the elements and consortia.

Because of the diversity of the SKA software, there will not be a one size fits all standard – there will be a layered approach and not all areas of the standard will apply to all software. Some things will be universal - all software will have to specify their dependencies; and some will be optional – for example use of a specific continuous integration environment. Some elements have indicated a desire to depart from the current waterfall-like model and adopt an agile approach to development, and part of the process is to define this agile process. Other elements may be best suited to a waterfall approach because they are closely linked to hardware. However, it is anticipated that the standard will have a significant impact in many areas, for example:

- SDP: Most sub-elements
- TM: Most sub-elements
- CSP: LMC. Possibly other software elements.
- LFAA: LMC and MCCS

- DISH: LMC. Possibly other software elements.
- SADT: Software in the timing system.
- AIV: Principal customer of the end products. Will also have their own scripts that need to be developed, versioned, tested and sanctions.

2 References

2.1 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] Applicable Document 1

2.2 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] Reference Document 1

3 Requirements

Proposing that that all consortia must adopt and be certified to and existing software engineering standard, such as ECSS/CMM/TSP etc. would be simple, but would not necessarily have the desired results. More important is to document a base set of processes and standards we can agree on, and then improve them when necessary as the system evolves. If this agreed set then maps easily to an existing standard process, and we could adopt that standard then this would be ideal, but it is not a requirement for us to move forwards.

As a starter, the following is an incomplete set of requirements that should be defined (Note: whether these are actually SKA L1-type requirements, or just serve to define a standard is debatable):

- An Agile development process, including:
    - A defined semi-continuous/iterative process of design, development, testing, delivery, integration and verification.
    - A defined long-term roadmap for each element's development goals. There is not that much difference between a road map and a project plan, except the roadmap focusses on a set of periodic goals (i.e. like Intel's tick-tock roadmap), which are tied into risk reduction and delivering functionality when it is needed – both to coincide with the main waterfall milestones of the non-software parts of the project, and the AIV milestones. The roadmap shall be harmonized across the system and based on the roll-out plan.
    - Regular reviews scheduled to coincide when each of the periodic goals are delivered where we assess the progress to date and adjust the roadmap for the future.
- A continuous integration system and set of processes to validate that the software meets basic QA standards, including:
    - A central, globally visible, set of repositories so that we can monitor the current state of code development.
    - A workflow that ensures a code review of all committed code, and pull requests accepted by authorised individuals.
    - Software simulations/stubs/drivers/mocks for all major interfaces to enable sub-system and system level tests.
    - Automated build on commit.
    - Automated test on commit – with unit tests having certain minimum coverage requirements as well as sub-system and system level tests.
    - Automated documentation generation.
    - Deployment scripts, which includes bare-metal deployment so automatically documents the full dependency list.
- A set of basic software standards
    - Defined software license (preferably and open source permissive license: e.g. Apache2).
    - Defined primary dependencies, for example:
        - Principal O/S (e.g. Debian)
        - Principal software frameworks (e.g. Tango)
        - Principal GUI frameworks (e.g. Taurus)
        - Principal source code manager (e.g. Git)
        - Principal deployment manager (e.g. Ansible)
        - Principal supported languages (e.g. C++/Python, Java, etc.)
        - Etc.

- A process of approving and recording additional, element specific, dependencies. This could include FPGA development tools, for example, as well as more generic software tools.
- Basic coding standards:
  - C++: (e.g. Follow Google C++ Style Guide, but with C++ Exceptions allowed/required)
  - Python: (e.g. Follow PEP-8, PEP-257 and PEP-423 guides)
  - Java: (e.g. Follow Google Java Style Guide.)

4 Process

The development of the process should, first and foremost, be viewed as a harmonization exercise, rather than centrally enforced. The SKAO requires standards to be adopted to ensure that the final system is well understood, of sufficient quality, and maintainable, but SKAO relies on the technical expertise of the consortia to provide the core of the expertise.

The process will start with the hiring of a consultant to provide short-term resource in the Office to support the development of the process and provide experience of similar developments.

Simultaneously, there will be a request to all consortia to provide the current state of their thinking about their processes. Some consortia have given presentations on this, and some have more formal documentation. Consortia would also be asked to nominate one (or possibly two) people who would be willing and able to participate in a harmonization team.

This information would then be reviewed in the office with the consultant and the consortia representatives consulted to determine the following:

1. Tools/methods currently in use in the SKA
2. Key drivers
   a. From the SKA Organisation
   b. From consortia/contractors
3. Discussion of the expected development lifecycle for SKA.
4. Consideration of existing standards and mechanisms that could be adopted.
5. Identification of specific driving requirements or conditions that must be addressed.
6. Roles and responsibilities: Who will be responsible for what in the main construction phase, what is the responsibility of the contractors, consortia and SKA office.

This would be supplemented by video meetings with the consortia singly or all together, and in discussions in and around SPIE and at the LMC Harmonization meeting and the scope of the Software Engineering Plan would be agreed.

After this, the development of the Software Engineering Plan can be broken down into segments, for example, Software Development could be dealt with independently of Integration and Verification and Validation. This allows work to be focused on a small amount of the development plan at a time so that reviews and discussions can be more focused and productive. For each segment of the Software Engineering Plan the proposed activities would be as follows:

1. Development of a Software Engineering Plan segment by the consultant, taking into consideration the inputs from the consortia and discussions with SKA team members.
2. The proposed Engineering Plan segment will be reviewed by SKA Organisation and immediate issues addressed.
3. The segment will then be released to the wider consortia for review and issues and areas of further work identified.

4. Discussion and resolution of issues. Given the (potentially) large number of contributors it is unlikely that all issues can be resolved within the scope of the contractor. Consequently, mechanisms should be put in place to ensure that changes are carefully tracked and addressed over a longer time period – and this may be transferred to new staff to be hired in, or seconded to, the office.
5. Update of the Software Engineering Plan segment. Ultimately this would become an applicable document to tenders for software development contracts.

Once all segments of the Software Engineering Plan have been addressed, the entire Plan would be subject to a final wider review and can be released and made available as an Applicable Document to the future tenders.

4.1 Timescales

Proposed timescales:

- 25 May: Circulation of proposal to SKAO Engineering teams [Done]
- 27 May: Discussion at EIT Progress Meeting [Done]
- June: EPM's to request current thinking and documentation from CPM's, with response by 20 June. [In progress]
- 20 June: Engagement of consultant. Evaluation of responses.
- 23 June: Presentation of the LSST Software Engineering Process at SKAO at 11:00, followed by discussion about how their ideas are relevant to SKA.
- 4-6 July: Discussion at LMC Harmonization meeting.
- July-August: Development of Software Engineering Process.
- October: Presentation of Software Engineering Process at Stellenbosch