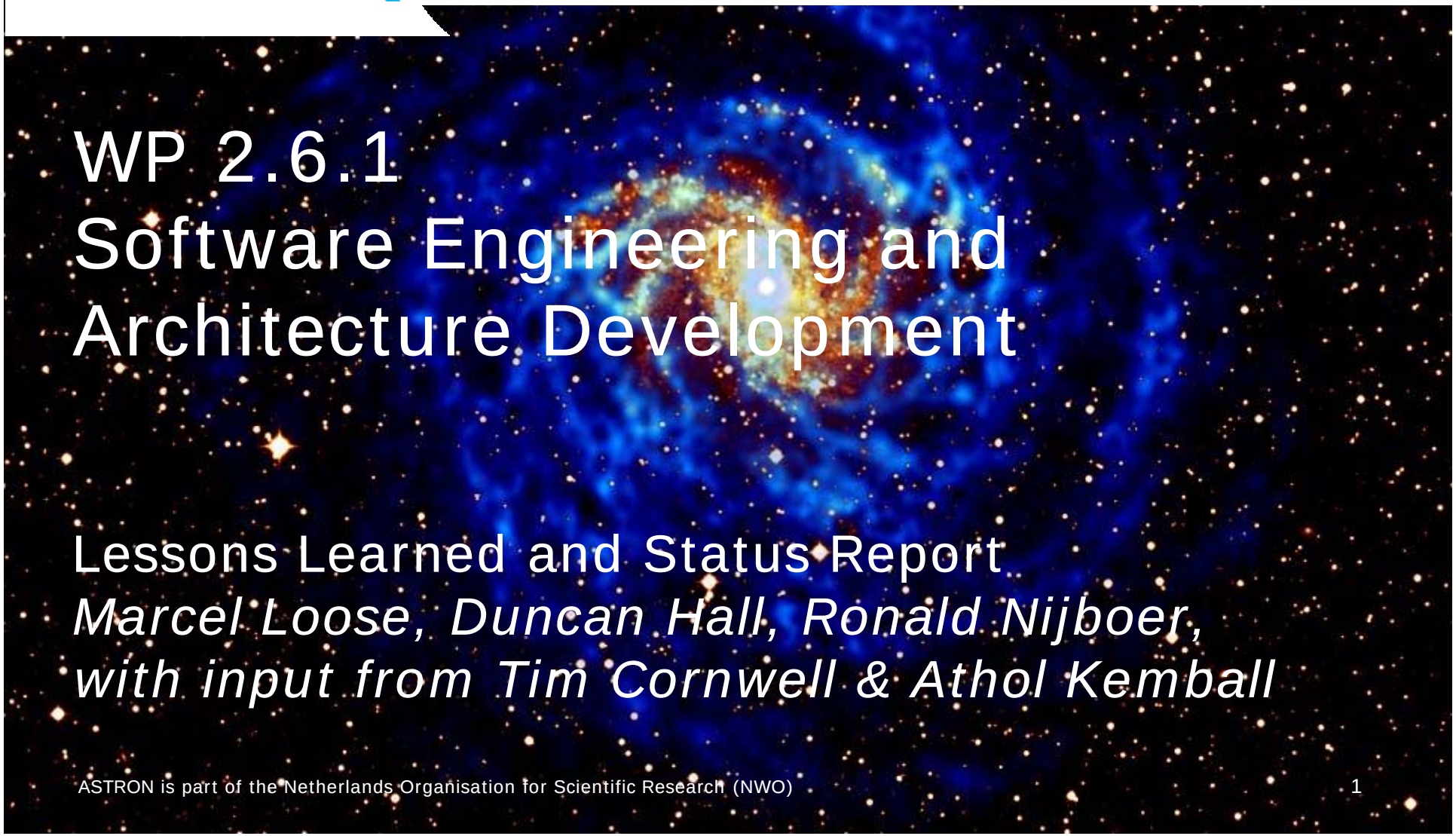

WP 2.6.1
Software Engineering and Architecture Development

Lessons Learned and Status Report

Marcel Loose, Duncan Hall, Ronald Nijboer,
with input from Tim Cornwell & Athol Kemball

Outline

- Lessons Learned
  - Challenges
  - Requirements
  - System Architecture
  - SWOT*) Analysis
- Software Engineering Good Practices
- Common Tools for Software Development
- High Level Software Architecture
  - Evaluate re-use of existing applications
  - Estimating the scale of software development
- Software Development Plan
  - WBS definition
  - Coordination of work activities
- Concluding remarks

*) Strengths, Weaknesses, Opportunities, Threats

Lessons Learned

Challenges

- The SKA will be immense in size, both in hardware and in software
- How do we manage
  - requirements analysis
  - system analysis & design
  - implementation and deployment
- Many different teams from around the globe might participate in software development
- How do we organize and coordinate these distributed teams?
  - Most teams are relatively small

Lessons Learned

Requirements

- Functional requirements are usually elicited from the domain requirements (i.e. the science and the operational model).
- Non-functional requirements are often overlooked.
  - Examples are scalability and extensibility
- However, non-functional requirements tend to have the largest impact on the system's architecture.

- Requirements analysis and requirements *tracking* are extremely important.
- Mapping requirements on the system's architecture is both a hardware *and* a software issue.

Lessons Learned

System Architecture

- It is *essential* to first create a good system architecture.
- The architecture is the *foundation* of the system.
  - Anything built on a bad foundation will break sooner or later.
- Iteratively *refine* the architecture, using the requirements.
- Create *loosely coupled* subsystems.
  - Define their *interfaces,* and *design* to these interfaces.
- Use well-known architectural design patterns and pattern languages.
- Design for change, because SKA *will* be built in phases.

Lessons Learned

SWOT: Strengths

- Large team of domain experts (i.e. astronomers)
- Prior experience, e.g.
  - SKA precursors
    - Meerkat, ASKAP
  - SKA Pathfinders
    - LOFAR, EVLA
  - ALMA
  - AIPS++, CASA, ...

Lessons Learned

SWOT: Weaknesses

- Too few real software architects & engineers.
- Academics tend to be headstrong.
  - “Not invented here” syndrome.
  - Conservative in use of modern development tools.
- Too little use of COTS software (libraries).
  - But there's so much – how to find/select what suits best?
- Recognition of need for formal processes
  - E.g., Lack of continuous unit and integration testing.
- Teams tend to work too much in isolation.

Lessons Learned

SWOT: Opportunities

- A lot of interest from industry.
  - Get them also involved in software development.
- Strong drive to succeed.
  - From scientists, engineers, and politicians
- World-wide effort, hence: A *lot* of knowledge
- Reuse of solutions already in use and understood
  - In radio astronomy, physics, COTS
- Learn from ASKAP and LOFAR
  - Useful documents can be found on the Wiki
    http://wiki.skatelescope.org/bin/view/SoftwareComputing/SoftwareEngineering

Lessons Learned

SWOT: Threats

- Huge system
  - Risk of losing ourselves in its complexity
- Mostly uncharted territory
  - Computing, network, (other) hardware
  - Algorithms, exploiting massive parallelism
- Very high expectations of hardware development
  - $O(10^4)$ increase of I/O bandwidth compared to LOFAR
  - $O(10^5)$ increase in computing power compared to LOFAR
- Lack of (use of) good tools
  - Requirements tracing
  - Analysis, design, implementation cycle
  - Domain Specific Languages (DSL) & Code generation

Software Engineering Good Practices

Contributions to date; Current Status

- Draft practices and procedures for software engineering [“Software and Computing Strategy”] have been published on the S&C Domain wiki for review
- As yet there has been no adoption of uniform software engineering methods

Challenges to be addressed

- Participating organisations with interests in software development activities have their own particular ways of working
- Practices, standards and procedures need to be defined to facilitate collaborative work on SKA software development
- Only a few early adopters

Software Engineering Good Practices

E.g., From the document "ASKAP Computing Architecture"

- Use hardware & software that adheres to *open standards*
  - This does *not* preclude the use of commercial products, but beware of *vendor lock-in.*
- (Re-)Use existing 3rd party software
- Don't over-design, refactor instead
- Deploy early and often
- Design to interfaces

Common Tools for Software Development

Contributions to date; Current Status

- Draft High Level Requirements for a Requirements Management Tool have been published on the S&C Domain wiki

Challenges to be addressed

- Participating organisations with interests in software development tools have their particular favourites
- As yet there has been no formal adoption of uniform software engineering tools

Common Tools for Software Development

ASTRON

- Benefit from expertise in non-core areas, e.g.
  - Packaging and distribution
  - System administration
  - Configuration management
  - Release management
  - Quality assurance
  - Technical documentation
  - Tool support
  - Development environments
  - Dependencies
  - Design
  - Reviews

(Chris Williams, SKA 2010 meeting)

Common Tools for Software Development

ASTRON

One exemplar model: CERN's ETICS

http://etics.web.cern.ch/etics/

- E-infrastructure for Testing, Integration and Configuration of Software
- FP 7 Infrastructure Project, originated by the Grid Community
- ETICS is a complete infrastructure for building, testing, configuring and managing distributed software projects

Common Tools for Software Development

High Level Requirements for Requirements Management Tool

General Criteria For Collaborative Tools	Description
1. Access control	• The tool must have access control whereby each participant has appropriate access to the data, e.g. identification and access control mechanisms for core groups and other user groups; identifiers for specific user teams to protect access to their specified, independent working spaces
2. Web access	• The tool must have a Web interface that makes it unnecessary to install a client application for occasional users
3. Multi-platform support	• The tool must support multiple operating systems, e.g. Windows and Linux
4. Usability, simplicity and customisation	• The tool should be easy to use • The tool should not require too much training • The tool should not require significant administrative effort • Deployment should not require extensive customisation o For example: take an unskilled computer user person order of 5 minutes to install; 5 minutes to be up and working on basic processes
5. Tool integration	• Integration to other tools should be supported through conformance to standards, e.g. o Import and export using XML Metadata Interchange (XMI), an OMG standard for storing UML models in XML format o Import and export to CSV or Microsoft Excel files o
6. Simultaneous use	• It should be possible for up to 10 RMS users to securely work on the same data at the same time
7. Information sharing	• The tool could support information sharing whereby all participants can be kept up-to-date, e.g. e-mail notifications, news groups, discussion forums etc.

Evaluate Re-Use of Existing Applications

Contributions to date; Current Status

- A review of documentation for several facilities that use solutions that could be re-used has been published for review on the S&C Domain wiki: "Towards SKA Software"

Challenges to be addressed

- The effort involved in integrating existing solutions is highly dependent on the specific details of interfaces and modes of operation
- These have not yet been defined for SKA1, or for subsequent Phases

IEAC: "Avoid in-house developments where adequate solutions exist"

High Level Software Architecture

Estimating the scale of software development required for calibration and image processing

Contribution to date; Current Status

- Sizes of current multi telescope / multi observing software codes for image processing have been researched – they range from several 100 thousand Source Lines Of Code (kSLOC) to >1,500 kSLOC
- Current codes do not offer “hands off” pipeline processing to achieve desired dynamic range of ~65 dB for SKA1

Challenges to be addressed

- It’s well understood that current codes can not simply be modified to provide high volume pipeline data processing
  - Precursors and pathfinders provide valuable experience, but scaling to SKA is non-trivial
- Good practice development of maintainable codes of size +1,000 kSLOC requires +1,000 tightly coordinated person years
  - Well defined standard data processing needed

High Level Software Architecture

Estimating the scale of software development required for calibration and image processing

Challenges to be addressed

- It's well understood that current codes can not simply be modified to provide high volume pipeline data processing
  - Precursors and pathfinders provide valuable experience, but scaling to SKA is non-trivial
- Good practice development of maintainable codes of size +1,000 kSLOC requires +1,000 tightly coordinated person years
  - Well defined standard data processing needed

Work to be done; Milestones; Risks

- Proposals will be developed for commissioning and initial science pipelines
- Further system development will be informed by lessons learned from precursors, pathfinders, large scale development of software systems for both industrial applications and scientific domains such as for the LSST and CERN

Work Breakdown Structure (WBS)

and work coordination within the software and computing domain

Contribution to date; Current Status

- Top-down planning and review processes – as defined in the PrepSKA Systems Engineering Management Plan – have been published on the S&C Domain wiki
- ~30 WBS elements have been defined and published on the S&C Domain wiki
- Lead organisations participate in monthly teleconferences; and by wiki contribution and email
- Lead organisations project manage their defined WBS areas

Challenges to be addressed

- Participating organisations have articulated challenges they face in adopting the industry-standard disciplines of top-down requirements-driven software development

Software Development Plan

~20 Organisations are listed as participants across ~30 WBSEs

Participant / WP area	ASTRON/LOFAR	AUT-CRS/AUT-IRASR	CSIRO/ASKAP	ICRAR	INAF	JIVE	JPL	KASI	NCRA [24]	NRC-HIA/DRAO	NRF/MeerKAT	OBSPAR
Cell colour Key:	Lead Organisation	Contributing Organisation	Referenced in [1]	Not in [1] 2.1.5 or 2.6.X	Expressions of interest	_	_	_	_	_	_	_
WP2.6.3.2 (3) Assess processing requirements for on-site and distributed data processing			Lead assumed as a result of Lead role							[1] Additional contributions from ... DRAO (PHAD) are expected		
WP2.6.3.2 (4) Determine the optimum use of real-time data from the SKA's monitoring and control sub-system			Lead assumed as a result of Lead role							[1] Additional contributions from ... DRAO (PHAD) are expected		
WP2.6.3.3 Aperture Arrays	Lead [1] ... is developing a new calibration approach for LOFAR.											
WP2.6.3.3 (1) Elicit and document requirements for Calibration and Imaging; extending Pathfinder and Precursor work	Lead assumed as a result of Lead role											
WP2.6.3.3 (2) Verify (demonstrate) and undertake improvements to calibration and imaging algorithms developed within the radio astronomy community	Lead assumed as a result of Lead role											
WP2.6.3.3 (3) Assess processing requirements for on-site and distributed data processing	Lead assumed as a result of Lead role											
WP2.6.3.3 (4) Determine the optimum use of real-time data from the SKA's monitoring and control sub-system	Lead assumed as a result of Lead role											
WP2.6.4: Non-Imaging Data Processing: - elicit and document requirements - formulate and document the overall strategy and approach to scaling - estimate implementation costs - algorithms for and demonstrations of data processing solutions - address risks and issues	[1] ... contributions from ... ASTRON ...	[17] CRS: Transient Radio Emission Array Detector Prototype ... Defining suitable and effective approaches and algorithms is necessary for the future radio telescopes such as SKA, LO-FAR and MWA	Lead [1] This task will be led by CSIRO (ASKAP) ...		[1] ... contributions from ... INAF ...	[1] ... contributions from ... JIVE ...	[5] The initial focus of this task involves adaptive detection and characterization of radio transients ... [27] ... algorithms that allocate computational and storage resources "on the fly" ... Reactive processing ... Automatic RFI excision ...		[16] ... there appeared to be some interest in the GMRT transient analysis pipeline as a prototype		[1] ... contributions from ... NRF (MeerKAT) ...	[30] I see two slots where I could potentialy contribute: WP2.6.4 ...
WP2.6.5: Data Products, Data Storage and Data Distribution: - elicit and document requirements - formulate and document the overall strategy and approach to scaling - estimate implementation costs - address risks and issues	[1] ... bringing in experience from operational ... European radio arrays.	[23] IRASR will contribute in the framework of IBM SUR Project: workflow and streaming of massive volumes of continuous radio astronomical data		[1] ... will contribute to all tasks [10] ICRAR's contribution to PrepSKA WP2 would be to the costed system design for an SKA data pipeline and archiving system ... [9] (IBM to) ... support ICRAR ...			[5] ... but they also have more general interest in techniques for archiving, mining, and managing large data volumes.		[16] ... CDAC expressed interest in contributing towards development / porting of pipelines ... scope for working together for some aspects of development of data handling and archiving tools, especially w.r.t. the developments being planned at ICRAR.			
WP2.6.6: Interfaces for Users and Operators: - elicit and document requirements - formulate and document the overall strategy and approach to scaling - estimate implementation costs - address risks and issues	[1] ... which has recent experience in developing operational models for the new-generation LOFAR telescope	[20] CRS: It might be possible to consider a contribution on other items in particular (15), (17), (37), (38) [20] CRS: It might be possible to consider a contribution on other items in particular (15), (17), (37), (38)	[1] ... which have operational experience with large arrays. [2] CSIRO is participating in the definition and development of the Pawsey Centre Petascale machine so we would like to contribute to the "Exascale computing and hardware" area.						[16] ... Proposal management system developed in collaboration with PSPL – now in use at the GMRT. Scheduling software developed in collaboration with TRDDC – being used for the GMRT.			
WP2.6.7: Exascale Computing and Hardware Extensive research, development and planning will be carried out ... to address the relatively low technical maturity. We include the demonstration of these technologies at SKA scale as an issue of technological readiness.												

Work Breakdown Structure (WBS)

and work coordination within the software and computing domain

Challenges to be addressed

- Participating organisations have articulated challenges they face in adopting the industry-standard disciplines of top-down requirements-driven software development

Work to be done; Milestones; Risks

- Lead organisations may be required to triage deliverables for each design review
- Once overall SKA1 system-level requirements have been identified and documented, S&C Domain requirements should be more easily elicited, documented and managed, including formal change control

Summary

- **Operational model drives software cost directly**
  - Pre-defined science cases
  - Defined modes of data processing (vs. PI driven processing)
    - Standard data products
    - Defined scope reduces risk
    - E.g. ASKAP surveys processing model, LOFAR pipelines
  - More use cases / flexibility magnify risks involved
- **Create loosely coupled subsystems**
  - Define and design to interfaces
  - Development teams per subsystem with few partners involved reduces risk
- **Other good practices**
  - (Re-)Use existing 3rd party software
  - Don't over-design, refactor instead
  - Deploy early and often