SKA Computing and Software
Primarily TMC, OSO, SDP, PSS, PST and related areas

Nick Rees
25 November 2019
Principle contributing countries
Outline

• Introduction and drivers
  • Non-functional requirements

• System wide aspects
  • Networks
  • Platform
  • Common software
  • DevOps and Agile

• Observation Management and Control (OMC)
• Science Data Handling and Processing (SDHP)
• Conclusions
Introduction and drivers
Software Overview
Why is software different?

• Non-functional requirements are critical
  • Essential for high quality and good architecture.
  • Difficult to specify, and often emergent.

• Software is highly modifiable and configurable.
  • Can evolve a system whilst continuously working on it.
  • Users can have different configurations of the same product

• Automation is relatively easy
  • Automated testing and integration systems change development dynamics

• Locality is often no longer an issue
  • Tooling is now mostly “in the cloud”.
    • Our software is stored by one company, compiled and built by another, and tested by a third – all automatically and continuously every time any change is made
SKA nonfunctional requirements

• Also called “Quality Attributes” or “ilities”

• Particularly important for SKA are:
  • Maintainability
  • Modifiability
    • closely related to maintainability for software
  • Performance – compute and power
  • Scalability
  • Testability
  • Availability and reliability
  • Affordability

• These are primary drivers of software architecture
What is clear

• With a project the size of SKA, we needed a modern, professional approach to software.

• With a project as diverse as the SKA, we must standardize our software processes wherever possible.
  • We focus on world leading standards and processes.
    • These are more robust than home grown ones.
    • They also provide a ready-made expert community that can be consulted.
    • Training and books are immediately available.
    • Contractors are often familiar with them.
    • However, they can always be tailored for our circumstances.

• My over-riding mission is to deliver quality software
System wide aspects
Network layer

- Most physical networks based on a “Fat Tree” approach.
- Multiple independent paths between any two points.
- Allows maintenance on infrastructure without downtime.
- Logical networks layered on top of physical infrastructure.
- Exception is some parts of the signal chain where:
  - Multiple independent paths are not economic
  - Layered logical networks would be detrimental – impacting performance.
Example – Low Network Diagram
Platform layer

• Most software will be deployed as containers.
  • Current exception is PSS
• We use “Platform” to describe the computing infrastructure, up to the container layer.
• We plan to share much of the container orchestration software across many sub-systems.
• This unifies development and simplifies operations.
• Also gives n+m resilience for hardware.

Kubernetes Architecture
Data Models

- The SKA has a significant data complexity.
- Data models will not be static and so have to be able to evolve.
- Looking to use or develop standards, where possible (e.g. IVOA and MSv3)
System wide standards

• Have a fundamental standards document that applies to all software. Covers:
  • Licenses
  • Lifecycle support
  • Source code dependencies
  • License compatibility
  • Permissive open source preference
  • Sensible development processes
• Have harmonization guidelines such as the Control System Guidelines
• Have common approaches to common systems such as logging
Tango Control System

In short:

- Control system framework
- Based on CORBA and ZMQ
- Centralized config. database
- Software bus for distributed objects
- Provides unified interface to all equipment, hiding how they are connected and managed
- Used across many sites so well tested and scalable
SAFe®, DevOps and other processes

• We are using SAFe® to harmonize many of the development processes.
• Team, train and solution pages are on confluence.
• All work is tracked in Jira.
• Software documentation on developer website.
• Systems automatically tested on engageSKA platform in Portugal
• Looking to broaden testing to other targets and the Prototype System Integration platforms.
Why Agile? – Cynefin decision framework

- **Obvious**
  - Tightly Constrained
  - Sense-categorise-respond

- **Complicated**
  - Governing Constraints
  - Sense-analyse-respond

- **Complex**
  - Enabling Constraints
  - Probe-sense-respond

- **Chaotic**
  - Lacking Constraints
  - Act-sense-respond

- **Novel Practice**

- **Emergent Practice**

- **Best Practice**

- **Good Practice**

- **Disorder**

©Dave Snowden, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=33783436
Agile mindset shift

• Agile development requires a mindset shift:
  • Project approach ⇒ Product approach
  • Budget ⇒ Investment
  • PM responsible ⇒ Team responsible
  • Build things right ⇒ Build right things
  • Fixed Requirements ⇒ Goals and evolving intent
  • Single deliverables ⇒ Multiple evolving prototypes
  • Measure deliverables ⇒ Measure impact & outcomes
  • Individuals ⇒ Teams
  • Command & Control ⇒ Coach and facilitate

• On time, On cost, on Scope ⇒ Meet goals, minimise waste, satisfy customer

• Transactional contracts ⇒ Relational Contracts
Observation Management and Control
Sub arrays

• With an array telescope we have the opportunity (and requirement) to partition the telescope for various reasons. This can allow:
  • Observers to group components to simplify describing their observations.
  • Science operations to schedule to multiple observations simultaneously.
  • Engineers to repair components in a way that doesn’t impact observations.
  • Engineers to test upgrades on the sky without taking over the whole telescope.

• If you talk to scientists, they typically only think about the former
  • And this is borne out by existing telescope designs

• If you talk to telescopes, they see most gains from the latter two
  • And to achieve it, the architectural impact is significant
  • A major architectural feature to attain maintainability and testability.
Partitioning stages

- System is not partitionable
  - It is effectively a monolith and cannot be partitioned.
  - Changes can only be made with a global system restart.

- System is partitionable, but not in an operational sense
  - All parts are required for the system to work.

- System is partitionable, but there is a major impact if a part is disabled
  - Parts are so large removal of one has a major impact on capability.

- System is partitionable and can operate effectively if a part is disabled.
  - As long as disabled parts are passive, the system can operate effectively.

- System is partitionable and parts can be maintained during operations.
  - Any maintenance and testing does not disturb the operating system.
  - Ideally, system supports live testing of new features in parallel with operations.
Partitioning in pictures

System Controllers

Hardware Controllers

Hardware Resources

System
Sub-array control

Control and Monitoring System

- Scheduler
- Scheduler B
- Subarray A
- Subarray B

Element Controllers

Hardware Controllers

Hardware Resources

Dish

Correlator/Beamformer

Pulsar Search

Science Data Processing
Sub-Array Distributed State Machine

- All sub-systems transition at the same time
- Enhances scalability since detailed coordination is delegated to the sub-systems
Monitor and Control System

- Responsible for monitor and control of the whole telescope.
- Architecture allows sub-arrays of all types (different science observations, or science and engineering)
Alarm and Notification Handling
• Manages the science lifecycle
OSO Data Archive

- Database likely to be a distributed, partitionable, eventually consistent model
- At least three locations:
  - Two telescope sites
    - For low latency and resilience
  - One cloud site
    - For world-wide user access.
Planning and Scheduling

- Planning is primarily supporting operations on longer timescales
  - Helping determine when the best time to schedule maintenance, for example.
  - Classifying observations with multiple prioritization criteria.

- Scheduling is primarily selecting which observation to observe next.
  - Propose a primarily demand-driven scheduler to allow rapid change in priorities due to weather etc.
Data Processing
Science Data Processor

Connections Key

- Signal Chair
- Queue Pub/Sub
- Tango
- Coordination
- Storage
- WAN Data
- Database access

Exploring the Universe with the world’s largest radio telescope
SDP approach

- Main challenge is with imaging data
  - Visibility data has significant data parallelism
  - Radio astronomy workflow can be described as a series of tasks acting on and transforming the data
  - Suggests a task-based architecture with management of data dependencies
  - Tasks implemented as pure functions (referentially transparent)
  - Underlying data dependencies described via an acyclic graph
  - Common approach in Big Data software
    - Many packages implement such task-based processing on directed graphs with lots of features to manage load balancing, failures etc.
Frequency parallelism

- Widely used in current systems
Frequency and spatial parallelism

- Needs complex data distribution to implement
- Reduces FFT size, at the expense of data duplication.
SDP Workflows

• Receive
• Pre-processing
  • Flagging & Excision of known RFI
  • Flagging & Excision of unknown RFI
• Real-time Calibration
• Fast imaging
• Imaging
• Deconvolution
• Pulsar search
• Pulsar timing

• Standard Iterative Calibration
• Model Partition Calibration
• Instrumental Calibration
• Pointing Calibration
• Ionospheric Monitoring
• Single-dish mapping
• Engineering Workflow
• Simulation Workflow
Example workflows

**Deconvolution**

1. Transform to Image Domain
2. PSF Image, Residual Image
3. Locate Peak
4. Peak model, PSF subtraction
5. Residual Image
6. Converged?
   - Yes: Deconvolution sky model
   - No:
5a. Restore model
6. Restored Image

**Imaging**

1a. Invert Workflow
1. Visibilities
2. Visibility Operations
3. Grid Visibilities
4. griddedData
5. Merge?
6. Fast Fourier Transform (inverse)
7. Images
8. Merge?
9. Image Operations
10. Images

1b. Predict Workflow
1. Model Visibilities
2. Model Visibilities
3. Degrid Visibilities
4. griddedData
5. Merge?
6. Fast Fourier Transform (forward)
7. Images
8. Merge?
9. Image Operations
10. Images
11. Strong Components
12. Weak Components
13. Model Components
14. Model Images
SDP Hardware Design

• Three networks:
  • High throughput Ethernet for WAN
  • Low latency network for LAN
  • Management network

• Four “personalities” of server:
  • Receive: receive streaming data from the instrument
  • Processing: process data in the buffer into SDP products
  • Service: handle routine tasks, such as monitoring and control, etc.
  • Storage: buffer data
### SDP Sizing

<table>
<thead>
<tr>
<th></th>
<th>Low (CDR)</th>
<th>Low (Now)</th>
<th>Mid (CDR)</th>
<th>Mid (Now)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Batch Compute</strong></td>
<td>12.0 Pflop/s</td>
<td>7.9 Pflop/s</td>
<td>10.0 Pflop/s</td>
<td>3.8 Pflop/s</td>
</tr>
<tr>
<td><strong>RT Compute</strong></td>
<td>1.8 Pflop/s</td>
<td>1.8 Pflop/s</td>
<td>2.1 Pflop/s</td>
<td>2.1 Pflop/s</td>
</tr>
<tr>
<td><strong>Input Buffer</strong></td>
<td>22.4 PB</td>
<td>43.4 PB</td>
<td>18.4 PB</td>
<td>48.5 PB</td>
</tr>
<tr>
<td><strong>Hot Buffer</strong></td>
<td>23.0 PB</td>
<td>25.5 PB</td>
<td>19.5 PB</td>
<td>40.5 PB</td>
</tr>
<tr>
<td><strong>Output Buffer</strong></td>
<td>0.66 PB</td>
<td>0.66 PB</td>
<td>1.1 PB</td>
<td>1.1 PB</td>
</tr>
<tr>
<td><strong>Input Buffer Rate</strong></td>
<td>0.70 TB/s</td>
<td>1.35 TB/s</td>
<td>0.6 TB/s</td>
<td>1.51 TB/s</td>
</tr>
<tr>
<td><strong>Hot Buffer Rate</strong></td>
<td>6.90 TB/s</td>
<td>8.24 TB/s</td>
<td>5.85 TB/s</td>
<td>12.16 TB/s</td>
</tr>
<tr>
<td><strong>Output Buffer Rate</strong></td>
<td>210 GB/s</td>
<td>210 GB/s</td>
<td>340 GB/s</td>
<td>340 GB/s</td>
</tr>
</tbody>
</table>
Pulsar Search

• Searches the sky and parameter space for transients and pulsars, including binary pulsars.

• A highly efficient signal processing pipeline with a single workflow.

• Workflow algorithms optimized to target COTS hardware

• Planning to process 3 beams/node
  • 1500 beams in Mid.
  • 500 beams in Low.

• Strict power and cost constraints because of desert location (in CPF).
Pulsar Search

TM/LMC

Data from beamformer

Data Receiver

De-dispersion Buffer

RFI Flaggering

De-dispersion Transform

Single Pulse Detector

Single Pulse Sitter

Single Pulse Optimiser

Data to SDP

Data Transmitter

RFI Flagged data

Dedispersed data

Periodicity search buffer

Fourier Transform

Birdie Zapping

Dereddening Spectrum

Fourier Domain Acceleration Search

Folded Candidate Optimisation

Candidate Sifting

Candidate Optimisation

Pulsar Candidates

Inversion Transform

Time Domain resampler transform

Fourier Transform and Power Spectrum

Harmonic Summation

Time Domain Candidate Optimisation
Pulsar Search Hardware and Network Layout

Data Network

Control Network

SKA Software, 25 Nov 2019
Pulsar Timing Hardware layout

- 16 servers spread across 2 racks.
- GPU accelerators in each node
- 1 beam/node
- One management server for pipeline control
Conclusions

• SKA will have a large, complex software system

• By its very nature, it requires different development and contracting processes than other parts of the system
  • We will be using agile processes and relational, not transactional contracts.
  • But they can all be seen in context within something like the Cynefin framework

• Quality attributes are very important
  • Has led to the development of the sub-array concept and scalability management

• The pre-construction consortia have created an excellent base for moving forwards.
  • But the design isn’t frozen and we look forward to developing them in the future.
Questions?
Lean, Agile and Scaling
What is Lean and Agile?

• The SKA has committed to:
  • “... overall the project/organisation is adopting a lean/agile approach”

• What is Lean?

• What is Agile?
Just so there is no confusion…

• Agile:
  • Is a structured process improvement framework
  • Uses the classic “Plan Do Check Adapt” Deming Cycle for iterative process improvements on fixed time intervals.
  • Optimizes the flow of delivered value based on lean product development principles such as continuously working products.
  • Controls costs by continual measurement of leading indicators and attention to business value, risks and time criticality.

• Agile is:
  • A highly disciplined engineering process which, when used well, has proven to be simply the best way to develop most software

• Agile is not:
  • a laissez-faire, best effort, no documentation process.
Most importantly

Lean and Agile are a mindset not a method or a process.

They are a way thinking, a way of looking at problems.

Be lean/agile rather than do lean/agile.
Product Development Methods

• Traditional, specify up front (aka “Waterfall”)
  • Best when requirements are well understood, with little ambiguity.
  • Simple example: Buying a Car or Computer

• Agile/Iterative development
  • Best when requirements are poorly understood, (or when system level optimizations are in play). e.g.:
    • Subtle non-functional requirements
    • Emergent requirements
  • Simple example: Developing a user interface for a new product.

• Research
  • Best when possibly not even sure what the question is.
  • Simple example: Understanding properties of new materials.
"If you only quantify one thing, quantify the Cost of Delay". —Don Reinertsen

### Agile vs Waterfall Development

<table>
<thead>
<tr>
<th>SIZE</th>
<th>METHOD</th>
<th>SUCCESSFUL</th>
<th>CHALLENGED</th>
<th>FAILED</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Size Projects</td>
<td>Agile</td>
<td>39%</td>
<td>52%</td>
<td>9%</td>
</tr>
<tr>
<td></td>
<td>Waterfall</td>
<td>11%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Large Size Projects</td>
<td>Agile</td>
<td>27%</td>
<td>62%</td>
<td>11%</td>
</tr>
<tr>
<td></td>
<td>Waterfall</td>
<td>7%</td>
<td>68%</td>
<td>25%</td>
</tr>
<tr>
<td>Medium Size Projects</td>
<td>Agile</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Waterfall</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Small Size Projects</td>
<td>Agile</td>
<td>58%</td>
<td>38%</td>
<td>4%</td>
</tr>
<tr>
<td></td>
<td>Waterfall</td>
<td>44%</td>
<td>45%</td>
<td>11%</td>
</tr>
</tbody>
</table>

“The overall results clearly show that waterfall projects do not scale well, while agile projects scale much better.”

Standish Chaos Report 2015

See: [https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf](https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf) The resolution of all software projects from FY2011–2015 within the CHAOS database, segmented by the agile process and waterfall method. The total number of software projects is over 10,000.
Why a Scaling Process?

• Agile implementations fail¹.
• The reality is their conclusion was;
  • “by attempting to work on multiple projects concurrently without a well recognised methodology for scaling up – many organisations are setting projects up for failure”
• We discussed this with the Software Engineering Institute from CMU. They had studied five alternatives:
  • Disciplined Agile Delivery (DAD)
  • Dynamic systems development method (DSDM)
  • Large Scale Scrum (LeSS)
  • Modular Framework for Scaling Scrum
  • Scaled Agile Framework (SAFe)²
• Our conclusion was that SAFe® was best suited to the SKA.
  • It is an adaptable and pragmatic approach that just encapsulates many lean and agile techniques.

 satu wasting £37 billion a year on failed Agile IT projects

¹https://6point6.co.uk/news/wasting-37-billion-a-year-on-failed-agile-it-projects/
²https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=484635