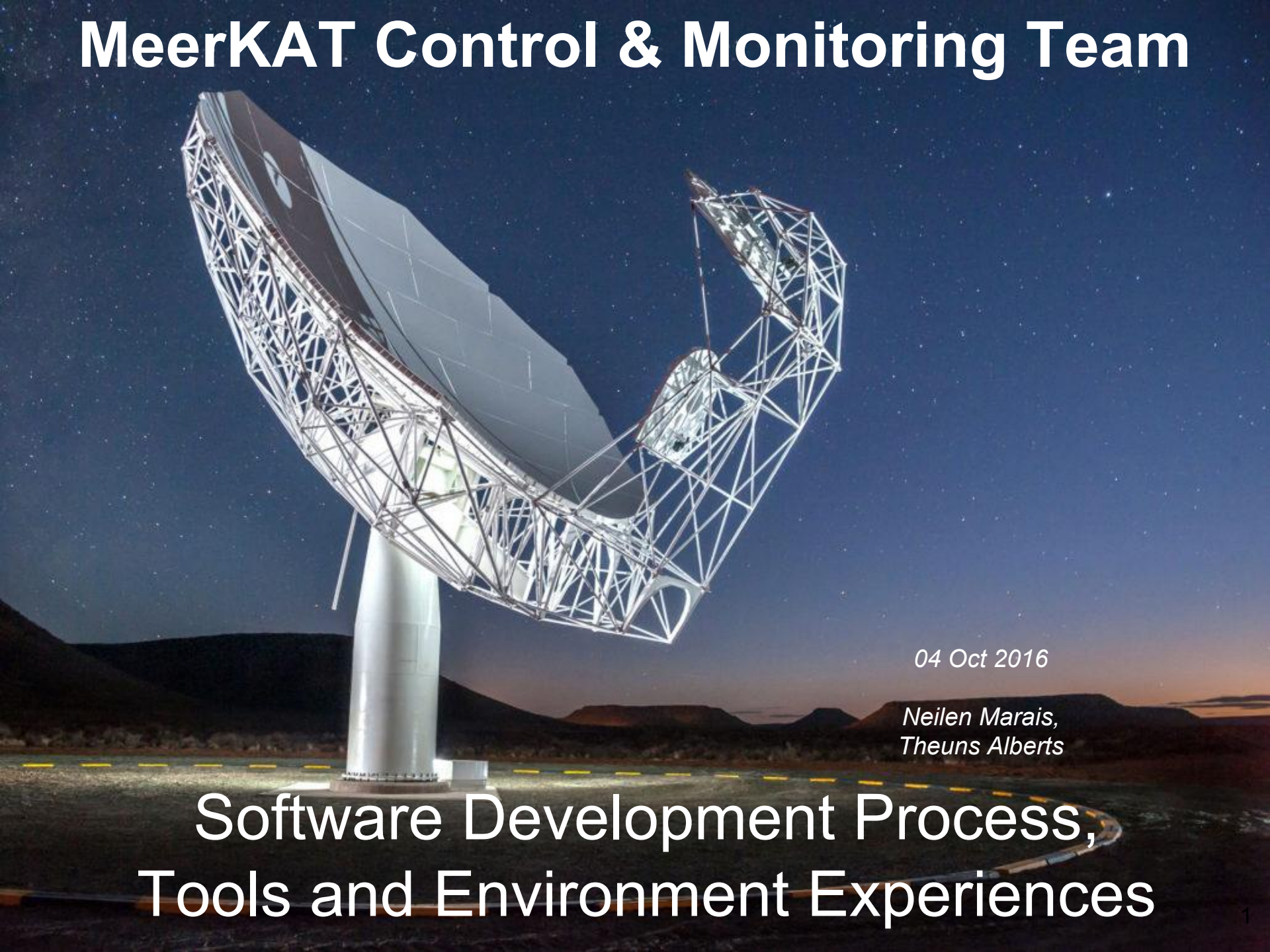
MeerKAT Control & Monitoring Team



04 Oct 2016

Neilen Marais,
Theuns Alberts

Software Development Process, Tools and Environment Experiences

Agenda

- Development Plan Overview
- Management Tools
- Development Process
- Development Environment
- GUI Development
- Documentation

Development Plan Overview

MeerKAT Project Phases

- Project split into a set of Array Releases [1]
  - Similar to SKA Array Assemblies

- CAM Requirements grouped into “Timeframes”. How?
  - Functional Analysis:
    - Requirements ... allocated to ... Functions
  - Project Planning:
    - Functions ... required by ... Array Release X
  - CAM Requirements Grouping:
    - Reqs/Functions for Array Release X … grouped as … Timeframe X

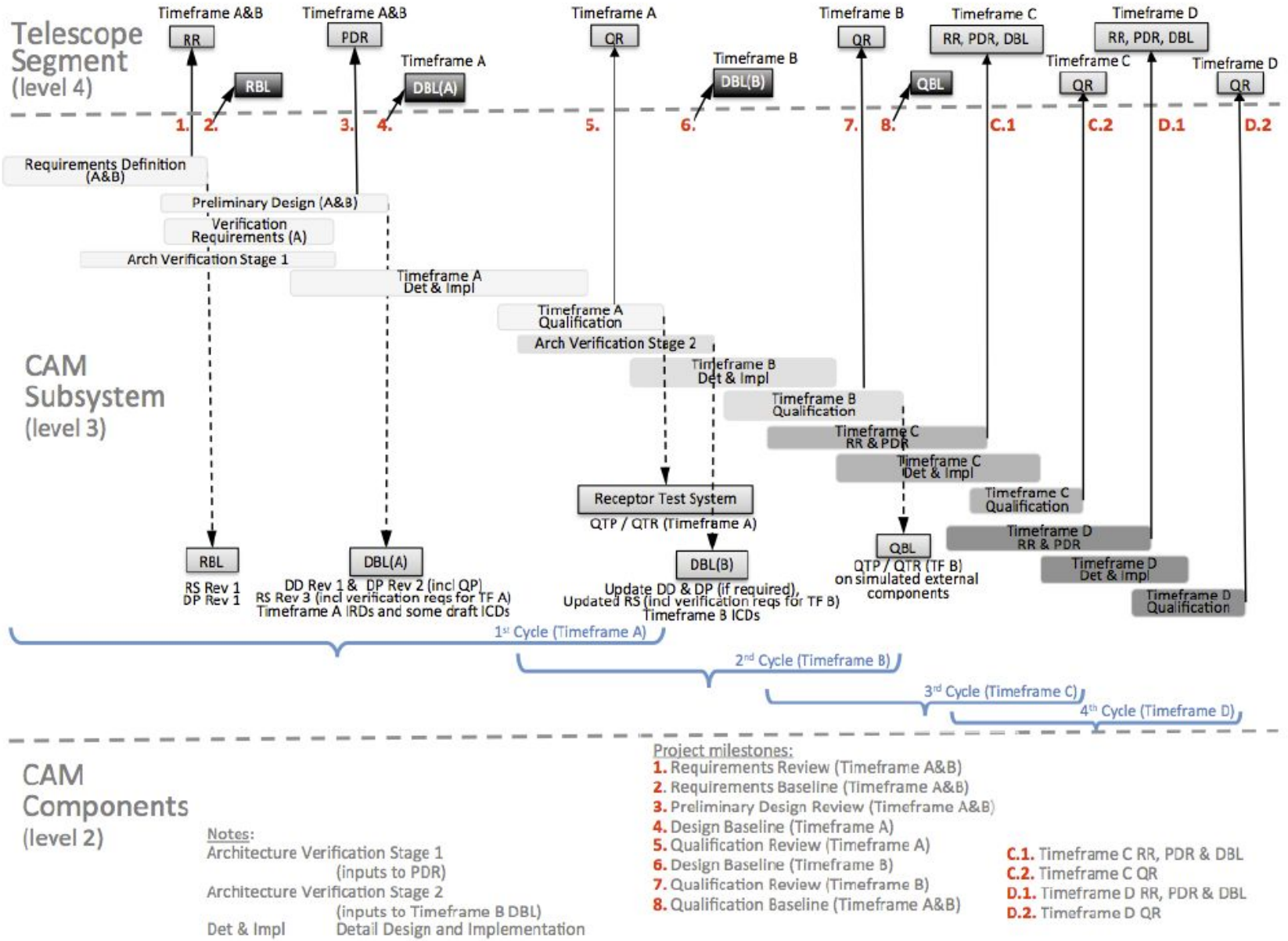
Development Cycles

- CAM knows exactly the required functionality and what requirements to verify and qualify against for each Array Release - this defines our Development Cycles

- Each cycle allows for:
  - Requirements review
  - Design review and baseline
  - Qualification baseline

- *Do not overlook the iterative nature and benefits thereof even for such large cycles at a high level of the project plan.*
  - ***but smaller is better***
  - RTS system suffered from spurious requirements

CAM Development Plan



CAM Components (level 2)

Notes:

Architecture Verification Stage 1 (inputs to PDR)

Architecture Verification Stage 2 (inputs to Timeframe B DBL)

Det & Impl Detail Design and Implementation

Project milestones:

1. Requirements Review (Timeframe A&B)
2. Requirements Baseline (Timeframe A&B)
3. Preliminary Design Review (Timeframe A&B)
4. Design Baseline (Timeframe A)
5. Qualification Review (Timeframe A)
6. Design Baseline (Timeframe B)
7. Qualification Review (Timeframe B)
8. Qualification Baseline (Timeframe A&B)

C.1. Timeframe C RR, PDR & DBL
C.2. Timeframe C QR
D.1. Timeframe D RR, PDR & DBL
D.2. Timeframe D QR

Development Experiences

- *Do not overlook the iterative nature and benefits thereof even for such large cycles at a high level of the project plan.*
  - ***But smaller is better!***

- RTS system had spurious requirements
  - Delayed rework of KAT-7 -> MeerKAT architecture
  - Wasted CAM dev time building MeerKAT-like features on KAT-7 architecture
  - Wasted correlator dev time trying to "look" like MeerKAT

- Commissioners not included early enough
  - Have incredible tacit information about early priorities

Development Recommendations

- Shorter SE cycles for software/gateware than hardware

- Be willing to modify ***system requirements***
  - For each cycle
  - On the basis of previous cycle(s) ***telescope user input***
  - Get buy-in from top level of project / SE
- Also: Ensure precise harmonization of technical terms used by subsystems as design progresses.

- Get minimal running prototype ASAP
  - Ties in with agile

- Idea: commissioners are "customers" in early phases
  - Commissioners should have clout
  - SE / top level project facilitate

Development Process

Agile

- An umbrella term for a set of methods and practices based on some core principles.

- Agile is not a silver bullet
  - can fail just like any other non-Agile project
  - it will allow you to fail faster though

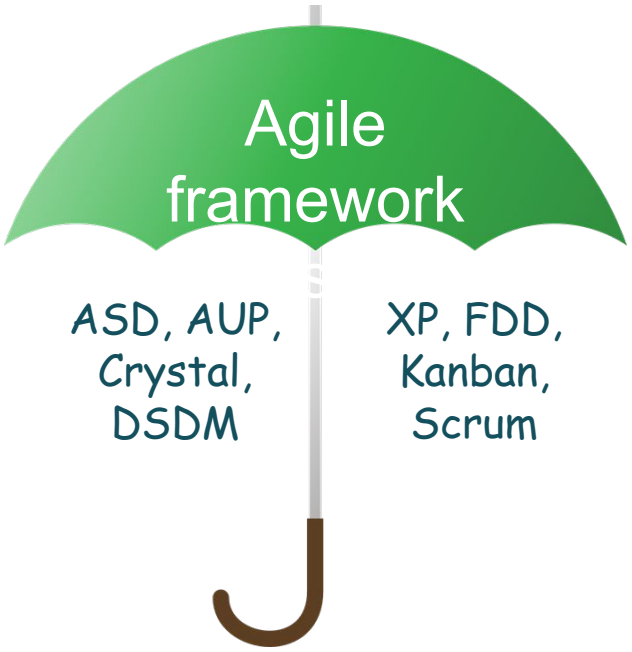
- Major benefit is the quick feedback due to the iterative development approach

- New? Take small steps, transition gradually and allow time

"You are Agile when you know enough about the practices to adapt them to the reality of your own specific situation."

Framework

- Various agile frameworks with various agile practices



- CAM uses Scrum and some other practices such as Continuous Integration

- Formally been using Scrum for about a year now

Scrum Practices

- Scrum Board etc: JIRA
- Two teams: 5-7 people, one team 3 people
- Iterations: ~~2 weeks~~ 3 weeks
- Keep to the Scrum events:
  - Sprint planning
  - Daily standup (timeboxed 15 minutes)
  - Sprint review (review work done)
  - Sprint retrospective (review the process)
- Planning
  - Timeframe Functions => Work Packages => Epics
  - Backlog grooming & Planning poker
  - Epics with combination of Tasks & User stories
  - 1 story point ~ 0.5 day

Scrum Experiences

- Note: Scrum is a ***process***
  - Every sprint is an opportunity to learn / tailor
- Light meeting load after sprint planning
  - Bad if you are in more than one team
- Developers and managers mostly positive
  - Project level SE adopted certain aspects
- Proper estimation is hard!
  - The more we do it the better we seem to get
  - Each iteration provides quantitative feedback
- Somewhat hard before some "product" exists
  - Try to get Minimal Viable Product (MVP) ASAP
    - Early and workable better than late and perfect
  - Include planning / spec generation in sprint
- Still unsure how to include experimental / speculative work

Practices: Continuous Integration

- Continuous Integration: Jenkins
  - Automatically triggered on user commit
  - Automates unit and component tests
  - Automates integration tests
  - Automates build steps
  - Automates container image provisioning [WIP]
- Jekins / Github integration [WIP]
  - Automatically run unittest for all branches
  - Only allow PR to merge if tests pass
- Automatic stable branch creation
  - When integration tests pass for RTS, KAT-7, MeerKAT
  - See Version Control slides

Continuous Integration Experiences

- Invaluable, but
  - A lot of work!
- CI must be treated as a product, just like the system it is for
- Need to be draconian about keeping tests passing
- Ditto test coverage, both unit tests and integration/functional
- Leverage CI tools
  - Automatic binary package creation
  - Automatic documentation / report generation
  - Automatic deployment image creation, etc.

- Basis of continuous delivery / deployment
  - Test deployment process too!

Development Environment

Environment

- Virtualisation: Proxmox
  - considering moving to LXD in future
  - Moving to distributed VM block device store (Ceph RBD)
  - OS: Ubuntu Server 14.04 LTS, will track latest LTS
- Messaging Protocol: KATCP
  - Zero tooling needed for basic debugging
- Programming languages:
  - Majority = Python 2.7+
  - GUI = Javascript (AngularJS framework v1)
  - some bit of C++ for device translators (with Python bindings using SIP)
- Databases: PostgreSQL & Redis (as in-memory store)
  - Monitored samples archive format: HDF
    - Moving to distribute object store (Ceph RADOS)
  - PostgreSQL Foreign Data Wrapper integrates archive

Version Control

- Software Configuration Management: Git
- Repository: github
- General workflow: All work done on git *branches*
  - Merge to up stream (e.g. *master*) branch via pull request
  - One or more persons review the changes in the pull request
  - Only merge after successful (approved) review
  - New github features make this easier to enforce
- Branch types:
  - master: bleeding edge of all merged development
  - stable: likely to work bleeding edge of all merged development
  - release: version as deployed to an instrument
    - hotfix: Fixes based on release branch
  - feature: long lived development effort kept out of master
  - user: short lived, workhorse branch type for most work

Version Control Experiences

- Git + github + modified gitflow works well for us
  - Much better than svn before
- Need to enforce use of VCS for all running code
  - NEVER leave uncommitted revisions running
    - Even for experimental fixes -> branches are cheap!
      - Never leave uncommitted changes around
  - VCS is software change control!
- Git tags provides strong mechanism for release control
  - Properly using them is WIP

Build Procedure

- Automatically kicked off by Jenkins when all tests pass
- Python Wheels are build for individual Python packages
- Debian packages for final software components [WIP]
- Packages uploaded to local repos
  - Using aptly [WIP] to manage repos for multiple releases / branches
- For formal version, software packages are git tagged
  - manual process
  - we use katversion
- *Future plan: Automatically build fully functional and qualified container images*

Testing & Integration

- Unit Testing: Python nose
- Component Testing: Python nose and mock
- Integrated CAM Testing: AQF
  - AQF = Automated Qualification Framework
  - AQF is a CAM implementation (nose based)
  - AQF allows integration tests to be written and tagged with the set of CAM verification requirements it implements
  - AQF produces a report with the result of all the integration tests (i.e. compliance of CAM towards its requirements)
  - Management really likes this

Testing & Integration

- Jenkins automatically runs:
  - both unit and component tests when a pull request is merged into master
  - integrated CAM tests daily against a fully simulated system (only the auto tests, not ones requiring demonstration)
  - Creating of stable branch if all system's integration tests pass
- Automated creation of Jenkins slaves
- Docker works very well for creating temporary test execution environments

Testing & Integration

- CAM Qualification Testing
  - Formal qualification testing by using the AQF
  - Includes demonstration tests with assistance provided by the AQF
  - Performed together with representative (usually System Engineering)
  - Performed on a fully simulated system (i.e. a virtual Karoo)
- CAM Acceptance Testing
  - Same as CAM Qualification Testing with a few unique tests
  - Performed in the Karoo on the deployed CAM subsystem with all other real subsystems
- Note good reuse of software integration tests + AQF for:
  - Day to day sanity testing
  - Continuous Integration
  - Qualification + Acceptance testing

Device Simulators

- Have simulator for every device/subsystem control interface
- Absolutely 100% invaluable
  - CAM Development before hardware / subsystems are ready
  - Day to day development / testing
  - Simulating error conditions
  - Integration testing
  - Demos
  - Interface clarification / development
- Even a very simple simulator can be highly useful
  - Most MeerKAT simulators use generic simulator library
  - Extended for more detailed simulators as required, eg:
    - MeerKAT AP simulator models antenna physics
    - CBF simulator can produce very fake science data

Deployment

- Legacy in-house deployment system that works
  - Question of timing (2011)
  - we've started to play around with Ansible
- Based on the Python fabric library
  - Use SSH based for application deployment or systems administration tasks
- Handles provisioning and software deployment
- Local *pypi* and *apt* repositories are used to speed up installation
- Aiming to deploy (or be deployable) monthly
  - Aims to be single-click deploy

Deployment Experiences

- Life without automated deployment is not good
- Deployment should be routine and regular
  - Easy to know what changed, Smaller changes
  - Quicker feedback
  - Better Change control
    - Can roll up / formalise hot fixes on short time scale
- Design for deployment
  - Support from CI
  - Support from virtualisation
  - Eliminate fear
- Remains non trivial

Other tools

- Messaging app: Slack
  - excellent integration with popular developer tools
- Developer machines:
  - dedicated development servers shared between developers
  - run dedicated containers for each developer
  - quick to deploy and destroy
- Code Analysis: Pylint
  - check against PEP8 standard
  - automatically run by Jenkins
- Code Coverage: coverage.py
  - not run automatically
  - done if there is a specific need or concern
- Computing Monitoring System: Ganglia

GUI Development

Operator UI Development

- Handled relatively informal
- Employed a “MeerKAT Telescope Operations Interface Control Document” (OICD) to capture user interface requirements
- UI requirements consist mainly of content descriptions and mockups of the required displays.
- Bi-monthly meetings were held with the relevant stakeholders (Operators, Commissioners, Engineers) to refine the OICD
- After few weeks the meetings became “live demo” sessions during which the displays were further discussed and refined

UI Technology

- Web Application
- Backend: Python Tornado
  - KATCP library uses Tornado event loop too
- Frontend: Javascript AngularJS v1
- Connections:
  - Monitoring: websocket
  - Control: RESTful API
- Security: nothing fancy
  - Basic Authentication (hashed password)
  - Role based Authorisation (session tokens)
- For more detail see [2]

UI Example (gif)

UI Technology Experiences

- Traditional Control System toolkits (e.g. Boy, CSS) not suited
  - High-level workflow based UI
  - Low level dials and knobs not that useful for telescope
    - Basic sensor displays + CLI control keeps engineers happy

- GUI implementation nominally "custom" but
  - Leverages widely used web frontend technologies
  - Wealth of JS libraries / widgets etc available
  - Easy to develop "custom widgets"

- Deployment and management advantages inherent to web technologies


Questions?

Documentation

Documentation

- Each software component has a:
  - Specification Record: Captures requirements addressed by the component
  - Design Record: Captures the detailed design of the component
  - API Description: Description of the classes, methods and attributes of the software component.
- Always available on-line
- Written in a markup language called reStructuredText
  - Documentation for Classes, Methods and Attributes are also written in same markup in the source code
- Compiled using Python Sphinx
  - automatically generates the API Description by pulling in the documentation from the source code

References

[1] CAM Development and Qualification Plan, Document Number: M1500-0000-001, Rev 5.

[2] The MeerKAT Graphical User Interface Technology Stack, ICALEPCS 2015, Theuns Alberts, Francois Joubert, online.