Square Kilometer Array - SDP: Architecture for Participation

a hackers perspective

catalyst
open source technologists

NZA

piers@catalyst.net.nz
Architecture for Participation

A combination of:

- Project management
- Solution architecture
- Project tooling

That will successfully support and utilise distributed project teams and resources
 Whilst still maintaining effective efficient use of resources
 ... and not sucking the joy out of life!
Square Kilometer Array (SKA)
Specifically SDP
SDP – primarily a big data problem

- SKA1-LOW
  - ~2 Petabytes

- SKA1-MID
  - 8.8 Terabytes
  - 8.8 Terabytes

- 100 Petabytes/year
  - ~5.6 Terabytes
Member countries highly distributed - time zones, accessibility challenges

10 member countries

9 interested countries

- Full members
  - SKA Headquarters host country
  - SKA Phase 1 and Phase 2 host countries
- African partner countries (non-member SKA Phase 2 host countries)
The (tiny) budget

Small compared to other mega projects:

- SKA1 SDP: Software and hardware ≈ €140M (approx. €90M for software)
- CERN = $CHF1.127B for 2016 (€969M)
- NZ IRD’s IT infrastructure and business process revamp = $NZ2.61B (€1.59B)
Construction – SDP – when?

- Best guess - due to start in late 2019

- Currently in pre-construction heading towards CDR – end of 2018
Project Management: what model to choose?

Too big and too unknown for traditional software delivery model:

- Detailed specification – not available for SDP
- Waterfall – requires full determinism
- Contract to prime – expects all expertise and resources to be singularly controlled, with fixed design and deliverables
  - Scope creep heavily punished
2014 – Open discussion of Failed IT projects by the Dutch Government on €4bn+ loses

“Just 7% of the projects with a budget starting at €7.5m can be said to be successful, Mulder told MPs. In total, 70% of projects fail. Of those which flop, 36% fail so seriously the new system is never used and around half are of doubtful value because they turn out to be too expensive, take too long or produce unexpected results, Mulder said. – source: DutchNews.nl” - Hans Mulder (professor University of Antwerp)

What did they have in common?

• They typically:
  • Out sourced to commercial IT partners
  • Fixed price on predetermined detailed design
  • Applied waterfall principles to project management
SKA preferred option: Large Solution SAFe
SAFe

- Adheres to basic Agile Development Methodology
- Constitutes the project management governance and rules – tends to hierarchical
- Optimised for large Enterprise environment – where there is leverage and control
- Leans towards longer release cycles
- Expects high degree of segmentation through solution trains at the programme level
The SKA – is different (sort of)

The Solution :-

- Capability of fundamental design to meet objectives is not certain – even after CDR
- Capacity of technology to support objectives is not certain – hasn’t it been fun for Intel!
  - Moores’ Law ending – no replacement insight

Unique circumstances in structure of SKA funding and participation model :-

- loose central authority, in-kind funding, multi-national, multi-institutional, low-commercial reward model
The SKA – needs a different kind of Agile (un-SAFe?)

To deal with :-

- Uncertainty – respond to evolving requirements as live testing and prototyping uncovers challenges
- Bleeding edge - requires specialisation and scarce resource
- Evolving participation – longevity means change – teams, team members, contributors will change over time
- Low commercial incentive to stay the distance
- multi-tier control structure, but fail-fast and bubble-up of prototyping impact required (loose and flat)
Solution Architecture for Participation

Modular design:
Solution Architecture for Participation

- Discreet component model
- Orient around knowledge domain as well as role
- Can it be developed independently without excessive integration dependencies (testing, delivery roadmap)?
- Interface versioning, backwards compatibility and deprecation – lifecycle management
How do you keep contributors involved long after the initial job is done

- Software components are for life not just for Christmas
- Someone has to maintain them, retain knowledge, improve, and integrate with changing context/requirements/innovations
- COTS can only do so much – evolves on different time scales, and to a different agenda
Project Tooling for the SDLC

- Develop integration standards
- Coordinate testing - automatic test gates
- Select infrastructure management solution that encourages participation – just like the public cloud!
- Uniform development tools, libs, packages, delivery formats, programming standards, issue tracking, governance, participation rules (conduct, deliverables, behaviours), test coverage, code collaboration tools
- Transparency – everyone can see everything – design submissions, meeting notes, code, testing, documentation
- Contributor induction, and support
- Reward
Tooling is fixed

- Tooling like culture is fixed early
- Invest in Infrastructure (including software defined infrastructure) early as this will not change for 10+ years
Test Gates

Continuous Delivery Pipeline

Continuous Exploration  Continuous Integration  Continuous Deployment  Release on Demand

Testing Gates

Single greatest control point for quality
What to do?
Crowd-source it

- Governance and guidance structure
- Clear architecture and interface design
- Modular and distributed
- Collaborative tool chain
- High quality testing and integration support
- Community driven

... in short, open source
What’s so special about Open Source?

- Incentives are primarily based on personal achievement and notoriety – there is (almost) no money in Open Source
- Darwinian – it has to be good, or it will die - build and they will come wont work (Google keeps re-learning this)
- Swarm and network effects – if something is hot then people gravitate
- Promotes fail fast – low cost and consequence experimentation
How it works in practice

OpenStack

• 6.8k code submitters, 30k participants, 107k commits/year (Feb, 2018)

• Governance: Board of directors, technical committee, user committee, then sub-projects

• Collaboration: Tools, resources, user groups, conferences, sponsorship, jobs, hall of fame

• Automated integration testing gates

http://activity.openstack.org/dash/browser/
In Summary: SKA needs

- Flexible scope – don’t change the objectives, but freedom to change the execution
- Lean and responsive - ability to fail fast and identify alternatives, with open communication
- Superb modularity and interface design – promotes component and team autonomy
- Automation of all the things
- A community oriented platform for participation - transparency
- Tight budget control – Accountability: reward on verifiable, on time, on budget delivery?
Square Kilometer Array - SDP: Architecture for Participation

a hackers perspective

NZA

piers@catalyst.net.nz
Architecture for Participation

A combination of:

- Project management
- Solution architecture
- Project tooling

That will successfully support and utilise distributed project teams and resources

Whilst still maintaining effective efficient use of resources

... and not sucking the joy out of life!

What are the key components for creating a solution and project architecture that will need to foster and leverage participation

A combination of:

- Project management practices
- Project support infrastructure and tooling
- Software and platform design principles

Architecture for Participation

=================================

As the SKA anticipates moving from a pre-construction phase to construction, focus is turning towards how a project on this scale can be managed efficiently, cost effectively, and successfully in an environment that will prove to be highly distributed both in terms of geography and team dispersal.

In order to achieve this, the SKA is looking increasingly towards modern project methodologies including the Agile principles of SAFe. However, there is an opportunity to learn from the successes that come from outside the traditional lines of comparison drawn such as large scale enterprise, and existing scientific and research endeavours. The new places to look for inspiration are the large scale open source projects such as OpenStack, Linux, Moodle, Python, Docker, GitLab where distributed participation, rigorous quality control, and routine and regular release cycles are the norm.

This talk looks at how these projects achieve their goals, and what SKA can draw from these experiences to build a successful collaborative, and enduring community project.
Introduce the SKA project, objectives, life span etc.

Specifically what is the SDP for and where is it located. How do the two implementations differ
- Galaxy Evolution (Normal Galaxies z~2-3)
- Cosmic Magnetism (Origin, Evolution)
- Cosmic Dawn (First Stars and Galaxies)
- Cosmology (dark energy, large scale structure)
- Cradle of Life (Planets, Molecules, SETI)
SDP – primarily a big data problem

Basic data flows
CSP to SaDT to SDP to Archive
The SKA organisation has ten member countries at present, with more to follow.
The (tiny) budget

Small compared to other mega projects:

- SKA1 SDP: Software and hardware ≈ €140M (approx. €90M for software)
- CERN = $CHF1.127B for 2016 (€969M)
- NZ IRD’s IT infrastructure and business process revamp = $NZ2.61B (€1.59B)

CERN is dealing with less data, and has a 1B/y budget
NZ IRD has a 1.6+B budget
We have 40+ M for hardware, and 90M for software for the complete build.

We also have a power efficiency cap.
Construction – SDP – when?

- Best guess - due to start in late 2019

- Currently in pre-construction heading towards CDR - end of 2018

Timelines:
Pre-construction - now
CDR – CSP done, SDP late 2018
Expressions of interest - TBD
Tender - TBD
Construction start – late 2019
Project Management: what model to choose?

Too big and too unknown for traditional software delivery model:

- Detailed specification – not available for SDP
- Waterfall – requires full determinism
- Contract to prime – expects all expertise and resources to be singularly controlled, with fixed design and deliverables
  - Scope creep heavily punished

The model for delivery will have to be radically different to make the limitations tenable.

- No regular IT will be able to deliver such complex software for so little money.

- Too big and unknown for traditional software delivery model
  - Detail specification
  - Contract to prime consultancy

No single prime contractor will be able to sign up to deliver this much complex software for so little money.
The model for delivery will have to be radically different to make the limitations tenable.
2014 – Open discussion of Failed IT projects by the Dutch Government on €4bn+ loses

“Just 7% of the projects with a budget starting at €7.5m can be said to be successful, Mulder told MPs. In total, 70% of projects fail. Of those which flop, 36% fail so seriously the new system is never used and around half are of doubtful value because they turn out to be too expensive, take too long or produce unexpected results, Mulder said. — source: DutchNews.nl” - Hans Mulder (professor University of Antwerp)

What did they have in common?

・ They typically:
  ・ Out sourced to commercial IT partners
  ・ Fixed price on predetermined detailed design
  ・ Applied waterfall principles to project management
What is Large Solution SAFe?

Scaled Agile Framework

AIV pipeline is equivalent to CI/CD
AIV manager is equivalent to Release Manager
SAFe

- Adheres to basic Agile Development Methodology
- Constitutes the project management governance and rules – tends to hierarchical
- Optimised for large Enterprise environment – where there is leverage and control
- Leans towards longer release cycles
- Expects high degree of segmentation through solution trains at the programme level

Tries to be a bit more formal – formal teams, formalised lines of authority and ownership
Uses longer release cycles
Expects greater control/certainty of design
The SKA – is different (sort of)

The Solution :-

• Capability of fundamental design to meet objectives is not certain – even after CDR

• Capacity of technology to support objectives is not certain – hasn’t it been fun for Intel!
  - Moores’ Law ending – no replacement insight

Unique circumstances in structure of SKA funding and participation model :-

• loose central authority, in-kind funding, multi-national, multi-institutional, low-commercial reward model

Moores Law is ending
Critical low level exploits are on the rise
The SKA – needs a different kind of Agile (un-SAFe?)

To deal with:-

• Uncertainty – respond to evolving requirements as live testing and prototyping uncovers challenges
• Bleeding edge - requires specialisation and scarce resource
• Evolving participation – longevity means change – teams, team members, contributors will change over time
• Low commercial incentive to stay the distance
• multi-tier control structure, but fail-fast and bubble-up of prototyping impact required (loose and flat)

Modern day software development practices preclude the waterfall method of detailed design up front. Too costly, to slow to ramp up, too uncertain. Requirements change before delivery because that is the nature of the business world. In last 20 years project delivery times have come down from years to months in terms of customer expectations.

To meet these challenges Business has had to change how it delivers IT. Tried outsourcing – it failed as the IP evaporated, and business lost visibility of how business rules codified in systems operated.

Moving to agile processes. Small continuous improvements, guided by high level strategic planning. Just enough design.
Solution Architecture for Participation

Modular design:

Components and abstraction is inherent in the design.
Solution Architecture for Participation

- Discreet component model
- Orient around knowledge domain as well as role
- Can it be developed independently without excessive integration dependencies (testing, delivery roadmap)?
- Interface versioning, backwards compatibility and deprecation – lifecycle management

Components defined at the right level to:
- Enable separation of work effort and teams
- Isolation from nasty surprises
- Enable retirement of technologies

Interface definitions, and technology selections:
- REST
- JSON
- Component roles
- OS
- Packaging

Life-cycle rules:
- Deprecation
- Feature introduction
- Backward compatibility
How do you keep contributors involved long after the initial job is done

- Software components are for life not just for Christmas
- Someone has to maintain them, retain knowledge, improve, and integrate with changing context/requirements/innovations
- COTS can only do so much – evolves on different time scales, and to a different agenda

COTs dictates a lot of design choices, and also dictates release cycles

- Security patching
- Feature releases
- Deprecation

Fork or die!
Project Tooling for the SDLC

- Develop integration standards
- Coordinate testing - automatic test gates
- Select infrastructure management solution that encourages participation – just like the public cloud!
- Uniform development tools, libs, packages, delivery formats, programming standards, issue tracking, governance, participation rules (conduct, deliverables, behaviours), test coverage, code collaboration tools
- Transparency – everyone can see everything – design submissions, meeting notes, code, testing, documentation
- Contributor induction, and support
- Reward

Continuous Integration and application deployment as a service
Required for:

Software Delivery Life-Cycle (SDLC) support for change management.
Software defined automated testing, and release management.
In order to support software development best practices, documentation, and traceability of software delivery it is highly desirable to implement automated software testing and release control.
Criteria:

Tightly linked to CI - the automated processes for spinning up test environments should be the same as for devs - consistency of platform. Need to be integrated with the master resource scheduler for the entire infrastructure landscape with a resource request interface with flavours to provide security/isolation from developers and testers
Part of development standards - automated code coverage reporting and test control gates
Coding standards enforcement - must handle automated running of source code quality testing tools such as flake8 and pylint3 - becomes part of the automated feature branch acceptance processes
Portable - burst to public cloud infrastructure or other institutions
Tightly bound to CI/CD - approved software gets packaged and pushed to the repositories, and the software defined infrastructure deployment processes pick this up
Migration/software refresh - automated testing based on upstream software updates for every aspect of the platform from the OS up.

Software repository services
Required for:

Source control, software packaging, container images.
The SDP construction project is to make use of and leverage open source software and
Tooling is fixed

- Tooling like culture is fixed early
- Invest in Infrastructure (including software defined infrastructure) early as this will not change for 10+ years
Test Gates

Single greatest control point for quality

Single greatest control point for:
Quality control – code, test coverage and compliance, documentation
If it doesn’t pass global compliance standards – it doesn’t get through
Catches issues of record at the point and time most likely to be corrected – ever tried to back fill code compliance, documentation and tests?
Test results are visible as are collaboration conversations
What to do?

Where can we learn from to ameliorate some of these issues?
Crowd-source it

- Governance and guidance structure
- Clear architecture and interface design
- Modular and distributed
- Collaborative tool chain
- High quality testing and integration support
- Community driven

... in short, open source

Follow the open source model of software delivery. Crowd source.

This is achieved through modularity, and distributed responsibility.

This is made possible through creating the right environment for participation. Everybody can talk to everybody else (Elon Musk! - one email away) and see what everyone else is doing.
What's so special about Open Source?

- Incentives are primarily based on personal achievement and notoriety – there is (almost) no money in Open Source
- Darwinian – it has to be good, or it will die - build and they will come wont work (Google keeps re-learning this)
- Swarm and network effects – if something is hot then people gravitate
- Promotes fail fast – low cost and consequence experimentation
How it works in practice

OpenStack

• 6.8k code submitters, 30k participants, 107k commits/year (Feb, 2018)

• Governance: Board of directors, technical committee, user committee, then sub-projects

• Collaboration: Tools, resources, user groups, conferences, sponsorship, jobs, hall of fame

• Automated integration testing gates

http://activity.openstack.org/dash/browser/

OpenStack – the poster-child. Largest open source project of all time.
In Summary: SKA needs

- Flexible scope – don’t change the objectives, but freedom to change the execution
- Lean and responsive - ability to fail fast and identify alternatives, with open communication
- Superb modularity and interface design – promotes component and team autonomy
- Automation of all the things
- A community oriented platform for participation - transparency
- Tight budget control – Accountability: reward on verifiable, on time, on budget delivery?