

# Agile Software Engineering and Systems Engineering at SKA Scale

Juande Santander-Vela,<sup>1</sup> and Luca Stringhetti<sup>1</sup>

<sup>1</sup>*SKA Organisation, Lower Withington, Cheshire, UK;  
j.santander-vela@skatelescope.org*

**Abstract.** The SKA is the ultimate software-enabled telescope, given the enormous amounts of computing hardware and software required to perform the data reduction. Systems Engineering is the set of processes and documentation required for managing and successfully realising large-scale engineering projects, but the classical approach is not a good fit for software-intensive projects, especially when the needs of the different stakeholders are not fully known, and requirement priorities might change. In this proceeding we give an overview of the system and software engineering processes in the SKA1 development, and the tension between classical and agile systems engineering, and how that tension helps progressing the design in a stable way, and ultimately leads to the successful construction of the SKA telescopes.

## 1. Introduction

The Square Kilometre Array (SKA) is an international project with the aim of building multi-purpose radio telescopes, with an equivalent collecting area of at least one square kilometre. It will provide unprecedented sensitivity and survey speed, so that key questions in modern astrophysics and cosmology can be answered. Currently, Phase 1 of the SKA (SKA1) is under design, nearing Critical Design Review (CDR) status.

To give an idea of the problem, data rates between the sensor network elements, and the processing elements of the SKA1 telescopes is shown in the figure below. The amount of processing required by the Central Signal Processor (CSP) is 50% of the current most powerful supercomputer in the world<sup>1</sup>, even if they are simple Multiply-Accumulate operations, and the Science Data Processor (SDP) will require in the order of 2.5 times the capability of the current most powerful computer in the world.

Eleven international consortia, whose members span 17 timezones,<sup>2</sup> are designing each of the subsystems. However, the SKA Organisation (SKAO) is the design authority, with Integrated Engineering Teams (IETs) per Element with Systems Engineering, Project Management, Operations, and Science considerations always represented. The External IETs include the counterparts from consortia, and we have Telescope Teams which can spawn Resolution Teams when issues need to be discussed and resolved across the whole project. SKAO retains ownership of the L1 requirements, System Architecture, System Baseline Design, and Functional Analysis, and after CDR we will

---

<sup>1</sup>98 PFLOPS as of June 2017, see <https://top500.org>

<sup>2</sup>from Vancouver (GMT-9) to Sydney (GMT+12)

have ownership of all system and subsystem requirements, and the SE and design artefacts from consortia. More information of SKA system engineering practices can be found in (Cremonini et al. 2016).

## 2. SE vs Agile development principles

Systems Engineering is an interdisciplinary approach and means to enable the full life cycle of successful systems, including problem formulation, solution development, and operational sustainment and use (INCOSE 2015).

During all of those phases, understanding of the system increases, and changes are inevitable. Managing change, and making sure change happens while keeping a working system, is central to SE.

However, in most of the typical SE processes there is the underlying assumption that change will decrease with time, which is typically true for hardware/mechanical systems, but tends not to be true for software heavy systems.

From the Agile Manifesto,<sup>3</sup> its main principles are:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Following those principles, the traditional Software Engineering Management Plan becomes more a framework for the best practices that need to be followed —pull requests between branches/forks versus commits to master, code reviews, Test Driven Design—, and not a final blueprint on how the software will be managed.

It also means that traditional Work-Breakdown-Structure oriented approaches don't fully work, and that instead, what is finally built is a result of agreements with stakeholders, as we will:

- Keep communication with all stakeholders involved in the project, with the conversation being
- To have continuous integration of the software systems
- To hire software developers (or development entities) that are invested in the system, not merely contracted; but also sharing gains with contractors if something is done faster than expected, or some desirable metrics are exceeded
- To have systems and processes that accommodate software change and still can prove requirements are satisfied; this means not using a document-centric requirements management system, but being able to integrated requirements, development, testing, and results with a quick turnaround.

And all of this is made on a fixed cadence —with the fastest period being the *sprint*—, trying to keep momentum, allowing teams to self-organise their prioritised tasks.

---

<sup>3</sup><http://agilemanifesto.org>

### 3. Marrying SE with Agile

Successfully marrying Agile principles and System Engineering principles means that, instead of making sure the SE processes are correct so that we remove the human factor as a source of mistakes, and that we can prove that the system, as designed, will behave as intended when built, we focus on delivering a minimum viable system as soon as possible, constructed from minimum viable subsystems, and we keep adding functionality, making sure we keep the intended behaviours that we have already built, and that we incorporate the still not incorporated behaviours as development progresses.

That means that we need metrics to keep track of the progress towards design completion during the design phase —the downward part of the classical V—, and then to record validation of the design artefacts —upward part of the V—. But you also want to be able to support those processes even when the design phase does not go through a traditional V, but it is made of multiple, indented Vs. See for instance Figure XXX, that reproduces Fig. 3.9 (INCOSE 2015).

If the killer function of Agile development is Continuous Integration, and be always available to deliver working software, the killer function for Agile Systems Engineering is Continuous Inspection, and be always able to deliver status of compliance, and of progress, almost at any moment, and on cadence.

### 4. SE functions during SKA pre-construction

During pre-construction, the main SE function from SKAO is the supervision of Interface Control Documents (ICDs), but also assessing design compliance against L2 requirements, through compliance matrices, and evidence for compliance. Many statements can be easily justified through functional allocations, while others require architectural support in order for them to be met.

Given the different levels of design maturity at system and element level, and of adherence to SE principles across consortia, and the relatively small amounts of SE personnel at SKAO, traditional processes with full flow-down from Design Reference Mission, to Operational Concept, to Functional Analysis, to System Requirements have not been fully possible. Rather, iterations over the available detail of those SE artefacts have been steadily increasing definition, with the drawing of Element and System PDR baselines, and future Element and System CDR baselines.

In all these processes, we have required submissions of some design and SE artefacts linked with individual milestones. SKAO has a unified system for Configuration Management, eBentley (eB)<sup>4</sup>, that is also used by all consortia. However, in spite of SKAO offers to Consortia for usage of our own Requirements Management system (JAMA<sup>5</sup>), it has not been standardised, and requirement submissions require manual processes from the SKAO SEs. There is still a large amount of human understanding required, and the process enables a less mechanistic, more engineering-centric review of requirements, but also more prone to pitfalls.

---

<sup>4</sup>eBentley

<sup>5</sup><http://jamasoftware.com/>

## 5. SE functions during SKA1 construction

The Scaled Agile Framework<sup>6</sup> (SAFe) has been selected as the way software development will be performed during SKA1 construction. SAFe allows for recursive groupings of Agile teams (using Scrum) into Agile Release Trains—that also use Scrum for managing the Train backlog, that then gets assigned to the teams’ backlog(s)—, and can group multiple Agile Release Trains into a Large Solution. See Figure XXX. As previously indicated, the System Engineering function will be organised from the SKA Inter-Governmental Organisation (IGO) that will exist before triggering construction. However, it needs to trickle down to the Agile Release Trains, and to the individual agile teams. Part of it has to go through the figure of the product owners, and consumer representatives, but there must be a strong presence of the system’s mission and behaviour, and the Operations Concept Document, so that it is really implemented.

We hope to be able to have more metrics, from the inspection of the backlogs, and the processing of the outcomes of the Program increment demos.

## 6. Conclusions

Traditional, *by the book*, Systems Engineering is never a good idea. It is even less ideal for projects that are optimising a multi-valued function in the cost, science capability, and time-to-construction, among other dimensions. Processes need to be tailored, and they need to be kept responsive, while at the same time keeping track of the System Engineering deliverables that are required as design artefacts, and managing the risk of the uncertainties that are still carried forward to the construction phase. When software is involved, design needs to focus on documenting the system architecture, instead of designing down to the application or module level, as agile methods will require continuous adaptation of the modules to be developed. It also requires definition and constant evaluation of the key performance metrics that will be used during the construction phase, and be able to translate between SE artefacts such as ICDs, the software architecture, functional analysis, and the needs for the different releases. We have been applying evidence-based System Engineering artefacts by different means, and found the shortcomings of the approach, specially when dealing with a world-wide distributed team with no traditional line of command.

## References

- Cremonini, A., Caiazzo, M., Hayden, D., Labate, M. G., Olgin, R., & Santander-Vela, J. 2016, in Modeling, Systems Engineering, and Project Management for Astronomy VI, vol. 9911 of Proc. SPIE, 99110W  
INCOSE 2015, Systems Engineering Handbook (Wiley)

---

<sup>6</sup><http://scaledagileframework.com>