Agile Delivery in a System-of-Systems Context

More than the Sum of Parts

Introduction

Such are the opportunities available from emerging technologies in the 2020s, that many thought leaders are beginning to talk of a 'Fourth Industrial Revolution', driven by AI, connected devices and data[1]. But progress toward these technologies has not been uniform, and some industries have found their engineering practices adapt less easily to this new world than others.

When industries turn 'smart', when that threshold of technical complexity is passed, it usually requires a radical rethink of engineering workflows and practices. When the aerospace and defence industries passed that threshold in the aftermath of World War II, SE was the result [2]. When software engineering passed that threshold in the 1990s, Agile Project Management was the result [3].

In the context of 'Industry 4.0', many new industries are realising their engineering faces a 'complexity gap': that in order to keep producing up-to-date products with reliability, efficiency and value, their less formalised or traditional engineering processes are no longer suitable. Industries which long since adapted to a smarter world become a source of inspiration for that transformation, and interest in both Agile and SE outside of their traditional domains has markedly increased.

Agile adoption, on the face of it, seems to have progressed further and faster than SE adoption: surveys report that Agile has now been at least partially adopted by more than 70 percent of organisations [4]. Yet, Agile seems in many ways poorly suited to engineering outside of a pure software context. Among the reasons for this are [5]:

- Agile release schedules assume product updates can be frequent and inexpensive, with low integration costs for new functionality.
- Agile practices assume the fabricator of the product is the same as the designer, and that engineering does not require the input of multiple domains of expertise.
- Agile system architecture assumes product value is primarily derived from the cumulative benefits of discrete features, rather than holistic properties of the system.

In scaled contexts, even in Agile's traditional software engineering domain, these limitations become significantly more pronounced, with no consensus yet on best practice. The Scaled Agile Framework (SAFe), Disciplined Agile Delivery (DAD), Large Scale Scrum (LeSS), and Scrum of Scrums are just some of the competing scaled Agile philosophies [6], and each have their drawbacks.

Nonetheless, when applied in suitable circumstances, Agile methodologies exceed the capabilities of traditional SE in terms of cost, speed and change readiness where individual subsystems can largely be treated in isolation and conform to clearly specified requirements.

By responding to complexity with an emphasis on a whole system view, SE has considerable potential to address these drawbacks. But on the face of it the two methodologies have many incompatibilities, including in their approach to documentation, decision gates, upfront planning and detailed design activity.

By drawing on our experience in System-of-Systems (SoS) engineering in a military tactical communication systems context, particularly Tactical Data Links (TDLs), as well as the growing body of literature on hybrid Agile SE methodologies [7], [8], [9], we propose an approach to combining the two methodologies able to retain the advantages of both.

Open Systems Enabled by Open Interfaces

The idea that an SoS requires a distinct conceptual framework and analysis toolkit from a mere systemhas found particular relevance and applicability with respect to TDLs. Although these TDLs support the exchange of digitised voice, their primary purposes are, for example, the automated exchange of sensor data and digital command and control.

In principle, their implementation is governed by NATO Standardisation Agreements and US Military Standards. Unfortunately, those standards are sometimes ambiguous, inconsistent, or even incorrect, and have to be used as the basis of platform implementations by nations with first languages other than English (with Americanisms!). Combined at times with local imperatives to override the standards, operational problems arise in which the platforms of NATO nations do not exchange information as expected. In other words, they have interoperability issues.

Although constantly evolving, the standards that govern SoSs like TDL networks persist as institutions far longer than the life cycle of the individual assets that comprise them. Yet many individual SoSs exist for far shorter timescales, and all will certainly be expected to change configuration frequently and rapidly, using existing assets that are autonomous both in terms of function and life cycle.

In the TDL domain, a consensus has emerged that the crucial factor in successful SoS engineering is a strong focus on interface standards. Constituent systems should be open systems, in the sense of conforming to defined interface management standards for their inputs and outputs, allowing the SoS to treat them as much as possible as a black box. As long as all components of the TDL SoS have a common approach to information exchange, TDL networks can be assembled quickly and securely, and the long-term evolution of standards can proceed without constraint from the implementation details of individual constituent systems with autonomous functions and very different life cycles.

SE best practice in this context strongly separates the engineering activity associated with defining the interface standards for the SoS from the design and mechanical details of black box constituent systems. In the context of a traditional SE V-model, the top half of the V and the bottom half of the V are essentially independently managed and governed, passing only standards and validation results between them.

So, in other words, we have concluded that we can take from SoS engineering the understanding that interfaces and component systems should be engineered independently, and we have in SE and Agile two different management approaches best suited to managing one of those two parts of the engineering project.

Comparing Agile Systems with Systems-of-Systems

Hybrid ASE techniques driven by this open system architecture have unlocked some impressive capabilities, demonstrated well by the Johns Hopkins Applied Physics Laboratory's Multi-Mission Bus Demonstration project [7]. This project was a successful attempt to produce a military space satellite to the size- and weight-restricted 'CubeSat' specification, which would allow the satellite to be launched more cheaply through 'ridesharing' with other payloads.

The project had very strict time and budget constraints, and required extensive research and development. As such, traditional project management wasn't going to be apt to the challenge.

A pure Agile methodology wasn't going to work, either, because of the extreme constraints on how the individual components of the satellite had to fit and work together. As such, the project did some initial SE-like activity at the outset to define compatibility standards, the interface with the spacecraft bus itself, and the external form, fit and function of the individual subsystems [8].

From there, scrum-like teams were assigned to work on the individual component subsystems, and were empowered to make incremental improvements to the design within the constraints assigned by the overall system architecture. These constraints could be modified if necessary, but only through a more SE-like top-down committee involving all of the subsystem scrum masters and the program manager.

The project was a deemed a success.

Comparably, in a TDL context, SE activity with respect to the SoS as a whole by necessity has to proceed without the ability to fully specify the architecture and design of the constituent systems. Constituent systems are developed in isolation from, and outside of the operational control of, those responsible for the interoperability standards.

Furthermore, the SoSs assembled from them – individual TDL networks created for a particular mission – are assembled ad hoc, have extremely short life spans, and evolve rapidly as new systems join and leave the network.

In other words, TDL SoSs share many of the characteristics of an Agile system, as defined by INCOSE [10]. A roster of drag-and-drop encapsulated modules is reflected in the TDL context by the individual TDL systems expected to be able to participate in the network. A passive infrastructure of minimal but sufficient rules and standards that enable and constrain plug-and-play operation exists in the NATO Standardisation Agreements and US Military Standards. The rapid evolution of Agile requirements, and the capacity to maintain continuous delivery and integration, make comparable engineering demands to those made by the process of assembling SoSs in the form of individual TDL networks ad hoc with respect to a particular mission.

As such, the challenges of implementing Agile practices in a scaled environment are highly comparable to the challenges of SoS engineering, and the techniques developed to manage the latter can be successfully applied to the former.

Collaborative Hybrid Agile Systems Engineering (CHASE)

The primary benefit of Agile project management methodologies is in the implementation phase of projects: indeed, in pure Agile, other activities are compressed to such an extent that they are almost entirely managed ad hoc in the course of development, with minimal documentation and bureaucracy. To quote the Agile Manifesto [11]:

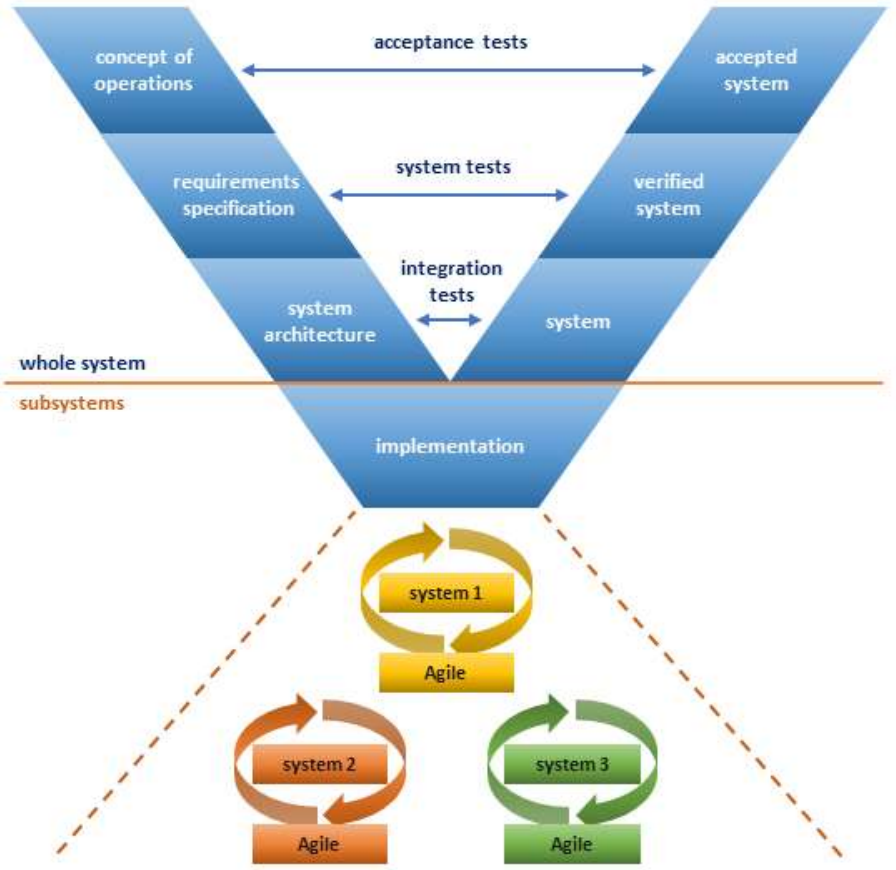
- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Most of the primary benefits of Agile practices, therefore, are derived from allowing them to manage their workflow according to Agile methodology with respect to their system of interest. For the sake of brevity, we will not review pure Agile methodologies in this paper, as resources describing scrum and other Agile methodologies are readily available.

Conventional scaled Agile frequently distinguishes between the 'project' and 'programme' layers of engineering activities, with the programme layer owning the programme as a whole, and the project layer usually being comprised of multiple scrum teams using Agile methodology to deliver individual subsystems.

CHASE methodology follows this distinction but proposes that the programme layer essentially be engaged in SE activities comparable to those employed in an SoS context: requirements management, modelling and architecture with respect to the SoS as a whole, but in which the output of the activity is not a detailed specification or design for the system as a whole, but rather simply interface standards with respect to the individual black-box systems of which the SoS will be constituted.

In measurable terms, SE activity should specify only the minimum system definition in which all requirements that affect more than one subsystem should be treated as pertaining to the interfaces. Inputs, outputs and purpose for all black boxes left in the model should of course be clearly and measurably specified.



CHASE V Model

From there, requirements for individual subsystems are handed to project teams to develop and implement, probably according to Agile methodology, though it should be emphasised that CHASE methodology allows the programme layer to be almost entirely agnostic about the methodologies used at the project layer. It should be emphasised that individual project teams need not all use the same methodology, sprint schedule or even be Agile at all: the relationship between the programme team and the project team in many respects may be thought of as more akin to a contractual relationship of acquisition and supply rather than a traditional relationship of hierarchical management, and as such, it retains an SE derived compatibility with a wide variety of development methodologies.

From the perspective of an individual scrum team, CHASE differs from conventional scaled Agile approaches in two ways: one more constraining, and one offering the team greater liberty. On the former point, scrum teams can expect slightly less freedom in their approach to implementation of their requirements, because of the need to conform to the strict interface specifications established at the programme level.

However, under CHASE, individual sprint teams have much greater freedom to choose their own work schedule. Because of the heavy emphasis on integration planning and standards in CHASE, the imperative for every project team to be running on the same sprint schedule, or indeed to even share a common Agile methodology, is greatly reduced. In this sense, CHASE is a much more decentralised methodology than most scaled Agile approaches, and can even be compatible with using entirely different project management methodologies at the project level, as well as subcontracting some requirements on a fixed price basis to an external supplier.

In common with most Agile approaches, project teams will be encouraged to practice iterative testing and continuous integration.

Subsystem and requirement verification will be run by the owning project teams on an ongoing basis, and work items will not be considered complete until compliance with interface specifications has been established.

In terms of requirements, although written down in a different format and structured differently as a whole body, at the most granular level, SE requirements and Agile user stories are more similar than different. Both prefer requirements to be specified in functional terms, atomically (only one objective per requirement), and with clear acceptance criteria. Nonetheless, because of the rigours of developing an SE model at the programme level, CHASE methodology employs SE requirements management practices, most notably in that they are arranged hierarchically, though this activity stops at a defined level, rather than proceeding all the way to the finest levels of detail.

Workflow management should, as far as possible, follow Agile in terms of work item prioritisation strategy, emphasising producing working functionality as quickly as possible. As such, the order in which work items are implemented will most likely be governed by an Agile-derived Kanban process.

However, when new requirements are added to the programme, they must first be integrated into the SE documentation and modelling before being added to the product backlog of the project teams. Depending on the release schedule for the project, it may be more appropriate for this activity to be undertaken iteratively using continuous engineering practices already familiar to the SE community, or, if more Agile practices are accessible in the product domain, on an ad hoc, as needed basis, adapting models and propagating revised interface standards as new requirements come in, with any necessary interface adaptations to existing functionality also added to the product backlog in a dependency-tracked fashion.

CHASE methodology is designed to provide a 'best of both worlds' combination of Agile and SE best practice, allowing the methodologies to coexist, and focus their activity on their respective domains of primary utility. For this reason, no process used by CHASE directly combines an SE process with an Agile process: what it provides is a framework for understanding which methodology is appropriate in which aspect of a programme. While CHASE likely involves both SE and Agile teams, which will operate largely autonomously, there are aspects which will require both kinds of specialist to understand and interface with processes associated with the other methodology. Nonetheless, although the processes are joined up differently, no individual process employed by CHASE departs from either the SE or the Agile canon.

By employing SE processes to specify interfaces between subsystems for the system as a whole, and employing Agile processes for the development of individual subsystems, CHASE provides the principal advantages of both approaches.

Dr John Hartas, SyntheSys.

Dan Hartas, SyntheSys.

References

[1] McKinsey.org. (2020). 'Industry4.0: Reimagining manufacturing operations after COVID-19'. [online] Available at: https://www.mckinsey.com/business-functions/operations/our-insights/industry-40-reimagining- manufacturing-operations-after-covid-19 [Accessed 7 May 2021]

[2] Hall, A. (1962). 'A methodology for systems engineering'. Princeton, N.J.: Van Nostrand.

[3] TechBeacon.com (2015). 'To agility and beyond: The history—and legacy— of agile development'. [online] Available at: https://techbeacon.com/app- dev-testing/agility-beyond-history-legacy-agile-development [Accessed 27 August 2021]

[4] McKinsey.org. (2020). 'The journey to agile: How companies can become faster, more productive, and more responsive'. [Online] Available at: https://www.mckinsey.com/business-functions/organization/our-insights /the-journey-to-agile-how-companies-can-become-faster-more-productiv e-and-more-responsive [Accessed 7 May 2021]

[5] Carson, R. (2013). 'Can Systems Engineering be Agile? Development Lifecycles for Systems, Hardware, and Software'. In: INCOSE International Symposium 23. INCOSE.

[6] Digital.ai. (2020). 'The 14th Annual State of Agile Report'. [online] Available at: https://stateofagile.com/#ufh-i-615706098-14th-annual-state-of-agil e-report/7027494 [Accessed 7 May 2021]

[7] Huang, P, Knuth, A, Krueger, R, Garrison-Darrin, M. (2012). 'Agile hardware and software systems engineering for critical military space applications'. In: Proceedings of SPIE Vol. 8385.

[8] Dove, R. and LaBarge, R. (2014). 'Fundamentals of Agile Systems Engineering – Part 2'. In: INCOSE International Symposium 24. INCOSE.

[9] Douglass, B. (2015). 'Agile Systems Engineering'. Waltham, MA: Morgan Kaufmann.

[10] INCOSE. (2015). 'Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, version 4.0'. Hoboken, NJ, USA: John Wiley and Sons, Inc.

[11] AgileManifesto.org. (2001). 'Manifesto for Agile Software Development'. [Online] Available at: https://agilemanifesto.org/ [Accessed 7 May 2021]

About SyntheSys

SyntheSys provides defence systems, training, systems and software engineering and technical management services over a spectrum of different industry sectors. Along with distinct support and consultancy services, our innovative product range makes us first choice provider for both large and small organisations. Established in 1988, the company focus is on fusing technical expertise with intuitive software applications to solve common industry challenges.