AGILE FOR HEDGEHOGS

THE FOX KNOWS MANY THINGS, BUT THE HEDGEHOG KNOWS ONE BIG THING

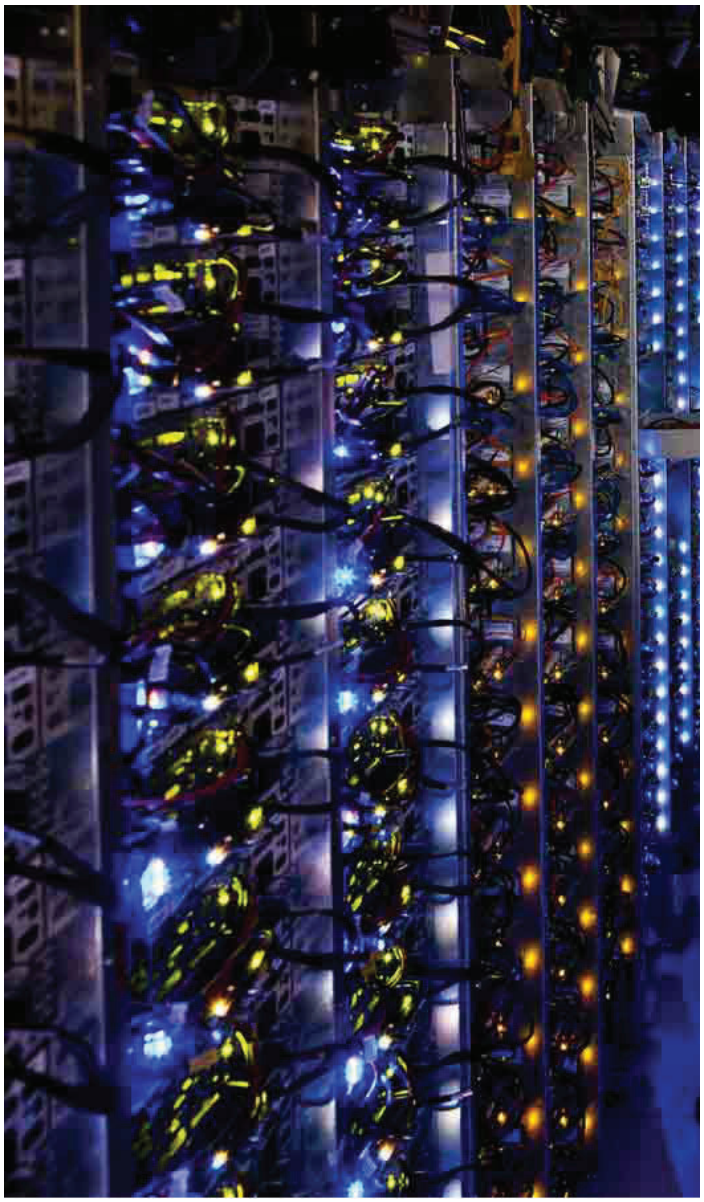
WHITE PAPER

SyntheSys TECHNOLOGIES

AGILE FOR HEDGEHOGS

A favourite quotation among philosophers of the last 100 years – inspiring essays, books and more – comes from the Greek poet Archilochus, and the fable of the hedgehog and the fox. It says: The fox knows many things, but the hedgehog knows one big thing.

In the fable, this perhaps refers to no more than the idea that for all the fox's cunning it is defeated by the hedgehog's one defence. In philosophy, this quotation is seen as getting at a deep difference which divides writers, thinkers and human beings in general; do you see the world in terms of a single unifying vision, principle or idea from which all your other thoughts and sentiments are derived, or do you see all sorts of different and diverse sources of value in the world, without ever trying to fit them into an all-embracing picture?



Agile software engineering was devised by, and to this day is to a large extent owned by, foxes.

In the 20 years since it was devised, it has remained a well-motivated approach to the challenges of developing many modern software applications. The 1990s were a time when the proliferation and complexity of software applications grew at an unprecedented rate.

Using traditional Project Management (PM) approaches, developing these systems could take years, and in that fast-changing world, by the time the software had been delivered, business needs had often already moved on.

In other words, software development had reached a complexity threshold beyond which traditional project management approaches were no longer suitable.

Agile's speed, adaptability and efficiency have led to widespread adoption across software engineering and even beyond in other fields, and any attempt to pick up the baton of meta-project management and take it forward must retain those advantages and do even more, because it's no longer possible to compete without them.

But... even now, Agile has not proved to be a panacea, and applied to certain contexts outside of software engineering, and even inside software engineering in certain environments, trying to apply Agile methods has failed in spectacular fashion.

Some other fast and adaptable approaches to meta-PM like systems engineering, which grew out of the aerospace and defence industry, have taken hold better in those contexts.

So pure Agile definitely has bounds and limits in terms of where and when it can genuinely be helpful, and there are times even in software engineering when a more hybrid approach could get you further, faster.

It's worth asking yourself, “would my product be better understood by the hedgehog or the fox?” In other words, are your customers deriving their value from lots of little things, or one big thing? Is your product valuable to them because of the cumulative benefits of discrete features, or because of the ways in which it is more than the sum of its parts; its emergent properties?

It's worth pointing out how much software has moved on from the environment in which Agile was conceived 20 years ago.

Back then, a lot of products were going in the direction of trying to be a Swiss army knife; loading in more and more features to become a one-stop-shop for any given activity, for example - how Microsoft Office was used back then.

Now, something like producing a document or even taking a rail journey will often involve using a wide variety of specialised apps that each handle a different part of the process for you.

Such a shift would have to retain the dynamism and efficiency of Agile software engineering, but improve on it by giving enterprises more freedom to choose in terms of large scale product architecture and long term strategy.

It would allow the emergent, holistic aspects of the user experience – which is so often all customers really care about – to be baked into the product from the ground up, and have it flow naturally out of the way your engineers work, rather than as a forced subject of constant vigilance, or worse, a coat of paint at the end.

And it would have to work in contexts where Agile has historically been more challenging to implement, such as at scale and under heavy regulation.

We aren't going to advocate a return to the traditional waterfall, nor a complete throwing out of Agile methodology in favour of something like pure systems engineering. But we think some cutting-edge projects have developed some hybrid-Agile approaches which are worthy of consideration even in a pure software context.

If you think your enterprise is more like a hedgehog than a fox, then these could be better forms of Agile for you.

AGILE AT SCALE

Even at the level of the individual Agile team, the world hasn't reached a fixed consensus about the best way to organise project management, but the competing approaches – Scrum, Kanban, XP, and others – are generally well understood.

They have advantages and disadvantages relative to one another, which allow them to be evaluated for the particular context. They also have a level of compatibility that allows them to pick up lessons from one another and mix together.

The same can't be said of approaches to taking Agile methods to the scale of an enterprise-sized project. A wide variety of approaches exist to getting multiple Agile teams to work together – Scrum of Scrums, Nexus, DAD, SAFe, LeSS – and other gesturing toward Lean and DevOps practices, tend not to have a great deal in common. They all, as anyone who has worked in such environments could tell you, also leave a lot to be desired.

We don't think anyone has got this quite right yet.

Agile came into existence because software products had become too complex for traditional project management methodologies to meet business needs. In software, the complexity threshold was reached when stakeholder needs evolved faster than the products engineers could build to meet them.

But there are all sorts of ways in which a complexity threshold could force you to move beyond a traditional waterfall; the breaking points for other industries have included capability, risk and cost. And other industries have responded to that complexity with equally diverse advances in their approach to engineering management.

The kind of complexity you face when you attempt to do Agile at scale is not the same kind of complexity that motivated Agile practices in the first place.

Agile wanted to move beyond delivering to fixed specification, cost and time requirements and instead emphasise a continuous delivery model that provided the highest priority working functionality as soon as possible. In other words, it was responding to a complexity that pressed on delivery speed. The complexity of Agile at scale isn't felt in speed per se, it is mostly felt in capability, quality and risk.

Overall product architecture becomes more important as the focus shifts more intensely onto dependencies, interfaces, integration and the emergent user experience. For hedgehogs – enterprises focused on doing one thing well rather than lots of things together – getting these aspects of the product right could represent a bet-the-business level of risk.

Because this is a different challenge to the one that Agile was originally designed to meet, it stands to reason that the solution might not so much involve extending Agile, as hybridising it with something else. Perhaps this could be a technique developed by a different industry, developed to move beyond traditional project management, but in response to a complexity challenge which more closely resembles the challenges of doing Agile at scale. And of course, it should be possible to synthesise them in a way that retains the best of both worlds.

Agile software development emerged in the early 2000s in response to the crossing of a threshold of technical complexity beyond which traditional project management was no longer suitable.

But software development wasn't the only area to reach a threshold like that, not even the first.

"For hedgehogs – enterprises focused on doing one thing well rather than lots of things together – getting these aspects of the product right could represent a bet-the-business level of risk. "

In the aerospace and defence industry, that complexity threshold was reached decades earlier, in the Second World War, with the first signs of change appearing at Bell Telephone Laboratories. The US Department of Defense started to reevaluate their engineering management in the late 1940s, in missile and missile-defence projects, and the burgeoning techniques they were adopting were adapted by NASA for projects in the space programme.

The disciplines developed in the course of these projects came to be known as 'systems engineering', and became the standard engineering management approach in both the defence and space industries.

Pure Systems Engineering (SE) solves these problems in very different ways to Agile, emphasising many processes that Agile rejects. But that said, systems engineering activity has many of the same benefits as Agile project management when compared to traditional engineering management approaches: better scope control, being better able to cope with changing requirements, more effective supplier relationships and introducing a single source of truth to the project.

Research has shown both disciplines to have dramatic effects on return on investment: 7:1 in the case of systems engineering activity[1], and 11:1 in the case of Agile project management[2], at least in their traditional industries.

Using SE techniques to help with scaled Agile doesn't have much impact on the individual Scrum; in fact, it couldn't, because at that level of granularity the approaches are largely incompatible. But at programme level and above, SE techniques can be deployed in combination with a fundamentally Agile approach to the day-to-day work of your development teams.

[1]Eric Honour. Systems engineering return on investment, PhD diss, 2013. University of South Australia. https://www.hcode.com/seroi/documents/SE-ROI%20Thesis-distrib.pdf

[2]David Rico, Hasan Sayani and Saya Sone. The business value of Agile software methods: Maximizing ROI with just in-time processes and documentation, 2009. FL: J. Ross Publishing. https://www.semanticscholar.org/paper/The-Business-Value-of-Agile-Software-Methods%3A-ROI-Rico-Sayani/e1f8a6a88a92b3c6f5cebb5ba0e50320f0e27115

SYSTEMS ENGINEERING

Like Agile, systems engineering developed its capabilities in response to the increasing complexity of the products that needed to be engineered. But because it evolved in a very different context, it achieved those goals in markedly different ways.

Probably the biggest difference between SE thinking and Agile thinking is that SE gives primacy to structure, and Agile thinking gives primacy to principles.

Put another way, systems thinking starts with 'how' questions, and Agile thinking starts with 'why' questions. Systems engineering is concerned with producing a comprehensive set of engineering processes of universal application (though only the most complex projects, if any, need adopt them all). Whereas, Agile project management is better encapsulated as a set of values, not per se concerned with particular processes (though of course some Agile processes have become fairly standardised), but rather encouraging engineers to design processes ad hoc to meet the needs of a particular problem.

In other words, if Agile belongs to the foxes, pure systems engineering is how hedgehogs solve the same sort of problems.

Systems engineering techniques are fundamentally about finding ways to analyse, model and plan the behaviour of a system as a whole and in its context above and beyond the details of individual components.

By having a suite of processes and tools designed to model and anticipate the structure of a system, projects can have assurance from the start that the right thing is being built in the right way, and that the project will interact appropriately with its context. This drives down cost by reducing the risk of mistakes and unanticipated defects, while simultaneously driving up quality by tying engineering activity more closely to precisely defined stakeholder needs.

Like Agile, SE acknowledges that your objectives at the end of the project will likely have moved on from what they were at the start, but SE tends to handle this by iterative processes, and by using scientific rigour in determining where the ambiguities and gaps are in customer objectives, anticipating potential changes at as early a stage as possible, and building flexible plans and readily adaptable designs for when change does come out of the blue.

This is possible because SE processes are in many ways analogous to the scientific method, designed with a similar philosophy and applying similar, rigorous and objective, techniques to the engineering process as science requires. In this analogy, formulating requirements are treated with similar standards to generating a scientific hypothesis, modelling and development are like carrying out an experiment, and quality activities are about rigorous measurement and testing the hypothesis against the results.

If an SE process for generating requirements is followed, it is more immediately and specifically visible when individual requirements are not clear, verifiable, functional or minimal, as well as when they are together incomplete or inconsistent. As a result, the question that needs to be answered either by stakeholders or by engineering is specified precisely and robustly, significantly mitigating the risk of an unanticipated need.

At the same time, SE is good at generating requirements for a project when there isn't a possibility of addressing specific technical specifications with the stakeholder; in fact, stakeholder conversations should avoid technical details as much as possible. Instead, focus should be on the functional description of the product, the 'use case': a problem that needs to be solved, or an opportunity that they want to pursue.

The context and environment for the system – its basic inputs and outputs – should be understood as clearly as possible while the system as a whole is still being treated as a black box. Only then, in a systems engineering process, do engineers start to formally investigate the sorts of systems which could solve the stakeholders' problem. In terms of separating stakeholder objectives from their technical implementation, SE resembles Agile, but where it differs is that it works toward understanding these for the system as a whole, rather than piecemeal with respect to individual features.

As well as the direct benefits of specifying project objectives with such a high degree of precision, this approach to requirements also enables systems engineers to construct sophisticated models of products, which can anticipate many potential problems before committing to development costs. These models touch on every aspect of the life cycle, and are designed to predict the behaviour of a system taken as a whole.

A formal systems engineering model is built out of black boxes, taking inputs from users and the environment and outputting stakeholder needs, progressively defining the contents of the black boxes as the design proceeds. Until the finest levels of detail are reached, the model is not concerned with how individual components work, but rather with the structure of a system as a whole: the inputs, outputs and interactions of system elements. It's about recognising that the structure of a system, rather than the specifications of individual parts, are what determines its behaviour as a whole.

As such, the models are nearly always built from the top down, with the system as a whole taking inputs from users and the environment and outputting stakeholder needs. As requirements get clarified and detailed, the model progresses down equivalent layers of complexity, at each stage fundamentally treating subsystems and individual elements as black boxes that transform inputs into outputs.

As such, the benefits of a systems engineering model are not just confined to presenting a clear, coherent architecture to designers, testers and operators; it also allows the behaviour of the system as a whole to be anticipated prior to proceeding with development, and potential defects to be anticipated and addressed before many of the associated sunk costs.

AGILE SYSTEMS ENGINEERING

Systems engineering, in its pure form, can be, and until ideas about hybrid Agile Systems Engineering (ASE) emerged, almost always was, deployed to manage engineering projects down to the finest levels of detail.

Traditionally, you would have expected this to be a 'horses for courses' choice: choosing an Agile or an SE approach depending on characteristics of your product or environment. In the aerospace and defence industry, integration costs are extremely high, and if they were not properly anticipated, defects arising from unexpected feedback interactions between apparently independent components could at best require you to go all the way back to the drawing board, and at worst result in a deadly disaster.

Product value is primarily derived from the interdependent functionality of the system as a whole rather than the cumulative benefits of discrete features.

In other words, while the sorts of products Agile was developed to manage are like a Jenga tower – a lot of features can be removed before the tower falls down – SE's traditional products are more like a house of cards; pull one feature out, and the whole thing collapses.

But because of this characteristic of SE's traditional product domain, the kind of complexity at which most scaled Agile approaches struggle – dependencies, interfaces, integration, the emergent user experience – is exactly the kind of complexity that SE techniques were developed to manage.

Essentially, ASE is about using both of these techniques together by mostly using systems engineering to steer the enterprise engineering from the programme level and above, and using Agile at the level of the individual team.

Because systems engineering requirements management and modelling is top-down, proceeds in hierarchical layers, and treats anything below the complexity level you're currently working on as a black box, it is immediately compatible with using a different set of project management techniques at granular levels of development.

Furthermore, SE and Agile requirements (user stories), while not historically written in exactly the same way, are functionally very similar; both must be written in terms of stakeholder objectives rather than a technical prescription, and both have high standards for testability, requiring clear and measurable standards for when the requirement has been met.

ASE uses systems engineering techniques to define and develop a broad product architecture, with strict and clear standards for how the individual components will integrate and interface together and with the overall design constraints and objectives.

Specifically, it envisions a product built out of a roster of drag-and-drop, self-contained modules, and governs both a passive infrastructure of minimal but sufficient rules and standards to enable and constrain that plug-and-play operation, as well as an active infrastructure.

In a continuous delivery context, this includes things like DevOps practices and whatever feature prioritisation algorithms and tracking that will be used by the project as a whole. Features can be prioritised using Kanban or anything else; SE practice is fairly agnostic about this, since it was designed for relatively static products with high integration costs.

The architecture should be designed with the capacity to swap plug-and-play components out as the requirements evolve, with only a minimal or well-understood effect on the behaviour of the rest of the system.

An ASE project begins by generating as wide an angle on stakeholder needs as possible, and formulating that as high level requirements. Because ASE requires more up-front activity on defining architectural and interface standards than pure Agile – though nothing gets set in stone – this might require a wider angle, a 'wish list', as the architecture should be designed such that as much unanticipated change in requirements as possible can be handled without having to revisit the interface standards.

From there, like in pure SE, those requirements are used to build a model of the system, at each layer of granularity treating subsystems as black boxes, though unlike in pure SE, this activity stops before you reach the level of granularity of individual user stories or features, to allow Agile teams their accustomed, value-powering freedom to solve problems on their own. These models are used to define the architecture and interface standards, which is designed centrally and before launching Agile development.

When that does kick off, individual Agile teams are given a functional description of the feature to be developed in input/output terms, and the general constraints of the architecture definition and interface standards.

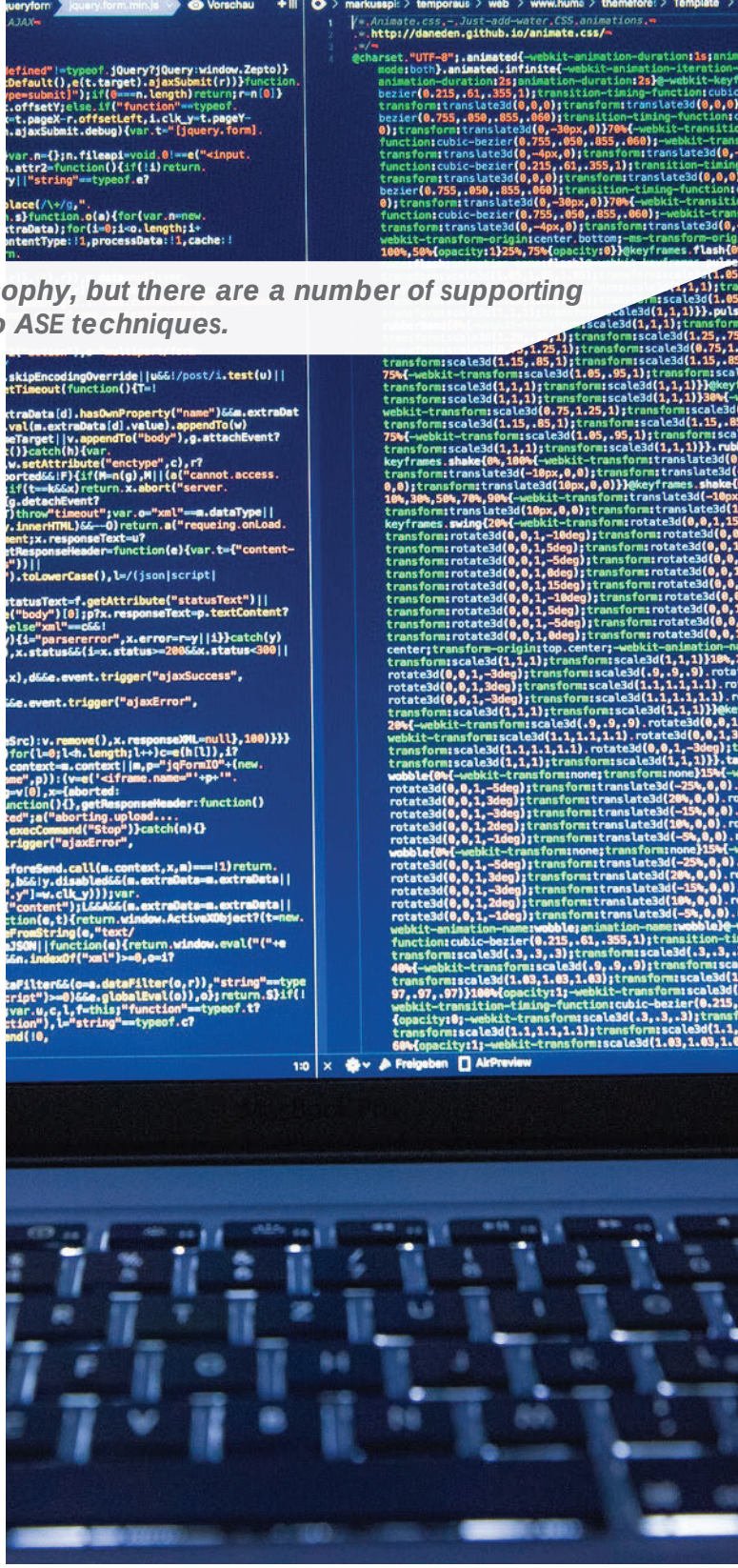
As a tradeoff for strengthening that constraint, it is worth pointing out that when using ASE there is no strong reason why individual teams need operate on the same sprint schedule (though of course it will be necessary for the programme team to be mindful of dependencies when assigning them features), the same Agile methodology or even use Agile at all, which among other benefits makes it much easier to work with external suppliers.

If the architecture or interface constraints do need to be modified because of emerging requirements, this falls within the purview of whoever owns the SE activity. Because those standards have been derived from an SE model, the implications of new requirements and how they affect the rest of the system can be quickly assessed and adapted to.

By using an SE model to provide a whole-system view, ASE simultaneously provides a new, very enabling perspective on the emergent, holistic aspects of the user experience for a product, while at the same time greatly easing the work involved with continuous integration, and allowing multiple Agile teams to work together with greater diversity of practices, without compromising the speed and adaptability of a pure Agile project.

ASE TOOLS

ASE is a bleeding-edge project management philosophy, but there are a number of supporting products and tools which can be readily adapted to ASE techniques.



Pure systems engineering strongly emphasises an integrated suite of project management tools, maintaining a single source of truth, and passing information seamlessly between requirements management, modelling, design, workflow management and testing.

Pure Agile, of course, tends to discourage reliance on sophisticated enterprise-wide software platforms, but for the SE aspects of ASE, centralised solutions are recommended. ASE projects probably don't want or need all of the end-to-end components of an engineering life cycle management solution, but there are elements of it to which SE tools are more appropriate.

One solution we provide is a partial implementation of the IBM® Engineering Lifecycle Management (ELM) suite.

In particular, Engineering Systems Design Rhapsody® is a gold-standard product for the SE modelling activities recommended by ASE good practice. The virtue of using ELM suite tools is that they provide a single source of truth and seamless integration between the different components and so the different stages of the development life cycle.

An Agile implementation of IBM® Engineering Workflow Management can help you translate these designs and models into a prioritised product backlog, user stories and release schedule, while providing transparency both within and between Agile and SE teams on progress, priorities and dependencies.

One solution we provide is a partial implementation of the IBM® Engineering Lifecycle Management (ELM) suite.

LET'S TALK

CALL:

+44(0)1947 821464

EMAIL:

CET@synthesys.co.uk

VISIT:

www.synthesys-technologies.co.uk

FOLLOW US ON LINKEDIN:

http://bit.ly/2ruSaWJ

FOR SALES, SERVICES & TECHNICAL ENQUIRIES

SyntheSys
TECHNOLOGIES