Enterprise Architecting White Paper

Integrating Enterprise Architecture & Agile Practices
TOGAF, User Stories and Use Cases

Michael Anniss

Contents

1 Introduction

This document describes in specific terms how The Open Group Architecture Framework (TOGAF), when applied properly, provides the basis for creating a flexible and responsive enterprise. In particular, it shows how it integrates with elements from the long history of approaches to agile development to create successful change at different levels of complexity, scope and timescale. It is a companion white paper to the more general paper on "Enabling Enterprise Agility" and provides a practical example of how to achieve such agility.

There are many different approaches to information systems development that have evolved over some 45 years and are today grouped under the broad heading of agile methods, such as; JAD (Lind 1974), Iterative, (Basili and Turner 1975), Rapid Prototyping (General 1980s), Evolutionary Delivery (Gilb 1988), Evolutionary Development (Crinnion 1991), RAD (Lowell Jay Arthur 1992), DSDM, (UK CCTA 1994), Scrum (Sutherland and Schwaber 1995), Fast Cycle (HP/MJA 1996), Unified Process (Booch, Rumbaugh, Jacobsen 1999), User Stories (Cockburn, Beck, Jeffries, Cohn 1998 – 2004), Extreme Programming (Beck 199), Agile Alliance (Many 2001), Agile Unified Process (Ambler 2005), SAFe (Many 2011), Disciplined Agile Delivery (2012).

TOGAF is structured so that it can be used together with any of these different methods as it can be tailored to any level of detail, frequency of delivery and level of risk and control and has always taken account of these approaches during its evolution. This paper provides an example of how TOGAF can be integrated with many of these approaches and can provide a well-structured and coherent context for delivering business services and information systems in support of an Agile Enterprise.

2 Enterprise Architecture and TOGAF

Enterprise Architecture is:

- A description of the elements within an organisation, what they are meant to achieve, how they are arranged, how they perform in practice, and how they respond to change.
- A framework (structure, approach and process) for managing change to those elements and their arrangement; to continuously adapt to organisational change in line with strategy (goals and objectives) and circumstances (specific requirements).
- The practice of acting to manage and evolve the Enterprise Architecture at all levels of control, change and pace.

The Open Group Architecture Framework, TOGAF is:

- A standard for a structure, approach and process for the effective management and control of Enterprise Architecture.
- A definition and description of that standard, used to plan, develop, implement, govern, change and sustain an architecture for an enterprise (see the TOGAF ADM - Architecture Development Method for the elements of that standard).
- A definition and description of the building blocks in an enterprise used to deliver business services and information systems (see the TOGAF Content Metamodel).
- A set of guidelines, techniques and advice to create and maintain an effective Enterprise Architecture.

TOGAF identifies the services required by an enterprise, the building blocks that define and implement those services; and the set of logical elements needed to understand the effectiveness of the changes to those services and building blocks. The TOGAF ADM and Content Metamodel provide the high-level framework for managing all levels of change within an enterprise and against which an enterprise, segment and capability architecture (see TOGAF levels - High level - Enterprise / Medium level - Segment / Low level - Capability) can be evolved through a set of specific solution architectures that implement change. Against this backdrop a delivery lifecycle approach is adopted to further generate the designs and implementations that deploy those architectures at relevant levels of detail. There will usually be significant interleaving between the change activities to ensure effective interoperation between the building blocks that deliver functionality, in many different situations, and at many different rates of change.

TOGAF identifies a series of logical steps (see the ADM) progressing from an initial understanding of a problem or opportunity to an implemented iteration of change; continuously responding to, changing requirements and feedback from earlier implementations. These steps can progress in a parallel or sequential fashion depending on the needs of each specific change for pace, risk, control and accuracy.

TOGAF does not demand any particular amount of detail or speed of change at each of these levels for all situations, it merely expects that an appropriate level of detail and speed of change will be adopted; based on the specific nature of the evolving architecture and the changes being requested (specifically in TOGAF this means selecting the relevant viewpoints, reference models and tools to satisfy the requirements being addressed and generate the outputs needed for any change).

3 Agile Techniques

Agile, when used as the label for the "agile approach to business and systems change", usually refers to how fast an organisation responds to opportunities and threats. It is typically recognised as the time in between an organisation becoming aware of a potential business opportunity or threat and successfully acting on it. Increasingly there is a demand for enterprises to deliver benefits more quickly. There is an imperative to respond quickly to drivers for change, and to be able to change direction promptly and safely. This use of the term Agile in the business arena has always been around but it is currently being specifically promoted in response to the use of Agile approaches for software development.

The Agile software development approaches identify how to develop and implement software in a particular manner, focusing on fast paced change and evolving architecture. They emphasise the use of small teams with shorter term delivery, bounded functionality, delivering a minimum viable product and fast evaluation of the emerging underlying architecture.

Many of the ideas that drove the development of Agile software development are found in the Agile Manifesto and the Agile Alliance and include:

"To provide guidance on how to create and respond to change and how to deal with uncertainty. You could say that the first sentence of the Agile Manifesto encapsulates the whole idea: "We are uncovering better ways of developing software by doing it and helping others do it." When you face uncertainty, try something you think might work, get feedback, and adjust accordingly. Keep the values and principles in mind when you do this. Let your context guide which frameworks, practices, and techniques you use to collaborate with your team and deliver value to your customers."

"One thing that separates Agile from other approaches to software development is the focus on the people doing the work and how they work together. Solutions evolve through collaboration between self-organizing cross-functional teams utilizing the appropriate practices for their context. There's a big focus in the Agile software development community on collaboration and the self-organizing team.

That doesn't mean that there aren't managers. It means that teams have the ability to figure out how they're going to approach things on their own. It means that those teams are cross-functional. Those teams don't have to have specific roles involved so much as that when you get the team together, you make sure that you have all the right skill sets on the team.

There still is a place for managers. Managers make sure team members have, or obtain, the right skill sets. Managers provide the environment that allows the team to be successful. Managers mostly step back and let their team figure out how they are going to deliver products, but they step in when the teams try but are unable to resolve issues.

When most teams and organizations start doing Agile software development, they focus on the practices that help with collaboration and organizing the work, which is great. However, another key set of practices that are not as frequently followed but should be; are specific technical practices that directly deal with developing software in a way that help your team deal with uncertainty. Those technical practices are essential and something you shouldn't overlook."

From Agile Alliance Agile 101.

4 Agility

Agility is usually defined as the ability to change (position, ideas, things) quickly and easily. It incorporates two main concepts:

- Stability of the structures that enable change to take place.
- Adaptability of the structures and their use in response to internal and external change.

Agility (in general rather than just the Agile approach) is a desired characteristic of an effective enterprise. It represents the potential to respond to changing situations and circumstances in an effective manner. Agility is based on a combination of stability (some elements that can resist threats while continuing to provide required structure and capability) and changeability (some elements that can adjust their structure and capability in response to opportunities, threats and change). Agility is the degree to which an enterprise can navigate an effective path towards successful adaptation between the elements of stability and change within its enterprise architecture. It must deal with stability and regularity just as much as change and emergence.

Agility is not just about small structures that continuously change rather it is the combination of a relevant set of properties working together:

- Balance - the ability to maintain effective equilibrium
- Static balance - the ability to maintain current structure over time
- Dynamic balance - the ability to maintain balance and change structure in response to change
- Speed of change - the ability to respond to change at an appropriate pace
- Strength - the ability to resist inappropriate change

Agility is not just the approach adopted by the Agile methods; that tend to emphasise the small, the fast and the flexible over the large, the slow and the stable. Real Agility is the combination of these elements into an organisational environment that not only responds to requests for change but also resists inappropriate requests for change by protecting its ability to continue to operate in line with continuing requirements as change occurs. An effective architecture is one that finds the most appropriate path between these often competing pressures.

5 Enterprise Architecture and Agility

TOGAF is about defining and evolving an architecture and implementing relevant delivery lifecycles; depending on the specific requirements for change and their impact on the evolving enterprise architecture.

TOGAF provides such a basis for identifying the changing structure and content of an enterprise and consideration of how to deliver that change using sound change management techniques (at a high level based on one of the three change lifecycle approaches identified later below). It also emphasises sound architecture and design principles such as preferring loose coupling and optimistic compensating integrations where possible but also considering strongly coupled and pessimistic controlled integrations where needed. Its foundational approach is:

- To understand the evolving strategic goals and objectives of the enterprise (Preliminary Phase and Meta iterations of Phase A).
- To identify the capabilities an enterprise needs to achieve its goals and objectives and develop an architecture vision (or sketch) that demonstrates the types of building blocks and the integrations between those building needed to provide those capabilities and deliver the required services. (Meta iterations of Phase A)
- To understand the requirements for each change programme and the nature of the change. (Requirements Management and Phase E, at meta and micro iteration levels).
- To then organise to acquire the building blocks and implement the services as quickly and easily as possible while ensuring that the building blocks address the specific requirements and the evolving goals and objectives of the enterprise and deliver the changing outcomes (Micro iterations from Phase A to Phase H).

Agile does not fully address the first three of these steps, rather it is one of the delivery lifecycle approaches that enables the fourth step. Agile is an approach and mindset for the delivery of specific solutions (somewhat limited by its focus on software change). Its focus on emergent architecture contributes to the first three steps above but that is just one element of managing a changing architecture and often misleading.

Agile says little about which specific building blocks are needed to understand the change (ABBs) or to implement the change (SBBs) (apart from Software, Dev Ops and some product/project management deliverables); merely that whatever is produced should be done within the agile change approach. The main building block identified in Agile is that of a physical software component (often approached from an extreme programming perspective). This is only one of many building blocks that may be required.

TOGAF therefore addresses a much wider set of issues than does Agile. It provides a framework for identifying and managing the enterprise architecture to support all business change (including information systems) across an enterprise and then provides the context for the delivery of specific solutions to deliver that change.

In a large enterprise Agile addresses how to deliver those specific solution changes implementing specific capabilities/functionality in an efficient manner but does not really scratch the surface of the more complex sub-system, system and between enterprise integrations and operations. However, in small organisations or isolated units within organisations, the Agile approach can support an element of architectural emergence as that architecture will be smaller, less complex and often subject to fewer constraints.

Like Agile, TOGAF emphasises adaptability and agility in all of these steps and has a foundational concept of just doing enough architecture to enable solution building blocks and business services to be implemented that meet the goals, objectives and requirements of an enterprise. In this sense TOGAF and Agile both start from the same underlying approach, do enough, implement in transitions based on value, gain feedback from what works in practice, review success and then respond to that learning.

	Rapid	**Agile / Functional**	**Robust**
Small scale	Y	Y	N
Large scale	N	N	Y
Simple	Y	Y	N
Complex	N	N	Y
Tried technology	Y	Y	Y
New technology	N	Y	Y
Stable business process	Y	Y	Y
Changing business process	N	Y	Y
Skills already in place	Y	Y	Y
Skills need to be developed	N	Y	Y
New solution	N	Y	Y
Extension of existing solution	Y	Y	Y
Integration of existing building blocks	Y	Y	Y
Integration of existing running services	Y	Y	Y
Short term change horizon	Y	Y	Y
Long term change horizon	N	N	Y
Time driven project (shorter timeframe)	Y	Y	N
Time driven project (longer timeframe)	N	N	Y
Cost driven project	Y	Y	Y
Quality driven project	N	N	Y
Function driven project	N	Y	Y
Structure/architecture driven project	N	N	Y
Low Risk	Y	Y	N
High Risk	N	N	Y

Table 1: The most common different delivery lifecycle change approaches and their essential natures

Note that for any project of some scale or complexity each chunk (sprint, iteration, stage, transition etc.) may require a different approach such that the project may well incorporate all of these approaches based on the specific circumstances).

Agile/Functional is one of the three main lifecycle delivery approaches and works really well in concert with TOGAF when used in a proportionate and controlled approach for delivering change, at an effective pace, and in response to the appetite for business change. However, there is both a faster type of approach, Rapid, and slower type of approach, Robust. The differences are shown in the table below. It is normally the case that any meta change of scale and consequence (a backlog of sprints in Agile, a series of transitions in TOGAF) will have a series of iterations of packages of work that will often involve all three types of change lifecycle. Also note that the Robust approach incorporates elements of Agile/Functional where appropriate and the Agile/Functional approach incorporates elements of Rapid where appropriate. They are essentially used in a nested manner depending on the complexity and scale of a specific change.

The three main lifecycle delivery approaches are:

- Rapid - Near immediate implementations of simple components (e.g. extended prototyping such as RAD)
- Agile/Functional - Fast cycles of component delivery for specific bounded functionality (e.g. JAD, DSDM, Agile)
- Robust (risk and architecture driven) - Longer term delivery of complex, large scale components, interoperable across the breadth of an enterprise or part of an enterprise (e.g. managed projects or programmes such as Prince 2 and MSP.)

In most change there will be a significant if not greater number of transitions implemented using Agile. Occasionally pure speed is needed (Rapid). In some changes really careful (Robust) approaches will be chosen when there is high risk or significant consequences associated with the change. When the Agile/Functional delivery lifecycle is the most appropriate approach it integrates well with TOGAF and the delivered sprints represent the transitions defined in TOGAF that move towards the evolving and changing target architecture.

5.1 Delivering the Right Units of Change (Products and Projects)

TOGAF has always defined a flexible approach in which specific outcomes and their value are identified and transitions are then created that deliver those units of value using sound project management practices. This approach aligns with the Agile concept of developing products/services over time through small and relevantly paced sprints, generating specific units of minimum viable product.

To facilitate this there are two TOGAF defined roles and another role that is part of wider business development practices. There is a general business development role of a Sales/Marketing Product Manager who decides what to do to address a market opportunity and how a product/service could be created and sold into specific markets.

TOGAF identifies a Solution Architect role (TOGAF Standard - Section 45.6.2 "The Solution Architect has the responsibility for architectural design and documentation at a system or subsystem level") of working with the Product Manager to develop the vision into a implementable concept and specify the relevant building blocks, their interoperability, and the target state for the product/service.

The Solution Architect then oversees the specification of the enabling architecture and the set of each of the minimum viable product transitions. The Project Manager identifies and organises the resources needed to deliver the enabling architecture and each of the minimum viable product transitions.

The Solution Architect and Project Manager then provide support and involvement as needed in support of the Product Manager for the maintenance and further evolution of the product/service over its full supported life cycle (taking account of any requirements changes and feedback from in-life experience).

These roles are performed as part of the development and delivery team and depending on the scale of the change may be partial roles or complete roles performed by more than one person.

5.2 The Right Scale

Agile has grown out of the Extreme Programming concepts of code led design, pair programming and co-location. These are all very useful tools in creating an effective environment for software development. However, these approaches are fundamentally wedded to small teams working within relatively restricted domains with limited or no sharing of state between those teams (as in the Bezos Mandate of 2002).

For enterprises who are following the Modern Design approach of designing and implementing stateless, loosely coupled micro components/services then relatively small scale is appropriate when:

- The functionality is limited to as small a unit as possible
- There is an avoidance of the complexities of strongly coupled, stateful and controlled components
- The limited functionality means complex large-scale integrated models (such as those required in major COTS products) are avoided.

This small scale does not necessarily work for large scale, integrity and risk driven solutions. While some of the building blocks of these more complex solutions can be built by small, isolated teams, the real challenge is usually not each team's localised architecture but the wider architecture supporting the integration between the many different elements at each level of system, sub-system and building block. The dependencies (often generating conflict between the specific requirements for and structures of each separate building block) and the integrations that together enable the end to end business service level to be delivered, require some set of overall planning and architecture to even start. The classical analogy of City planning provides some understanding of this.

- City level - provides the basic cross city services (such as roads, electricity supplies, water supplies, networks) in a way that can service all of the lower level segments. Provides integration with other cities, regions and countries as appropriate. Sets rules, regulations and standards for that city.
- District level - identifies differences in the nature of each district and specialises for that difference (such as industrial zones, residential zones, retail zones). Sets rules, regulations and standards for each district.
- Building level - identifies differences in building types, usually constrained by the district the building will be built in. Sets rules, regulations and standards for each building type.
- Mechanism level - where mechanisms run across all three types, they drive sharing within the context of each boundary. Sets rules, regulations and standards for each mechanism in concert with their deployment in each segment.

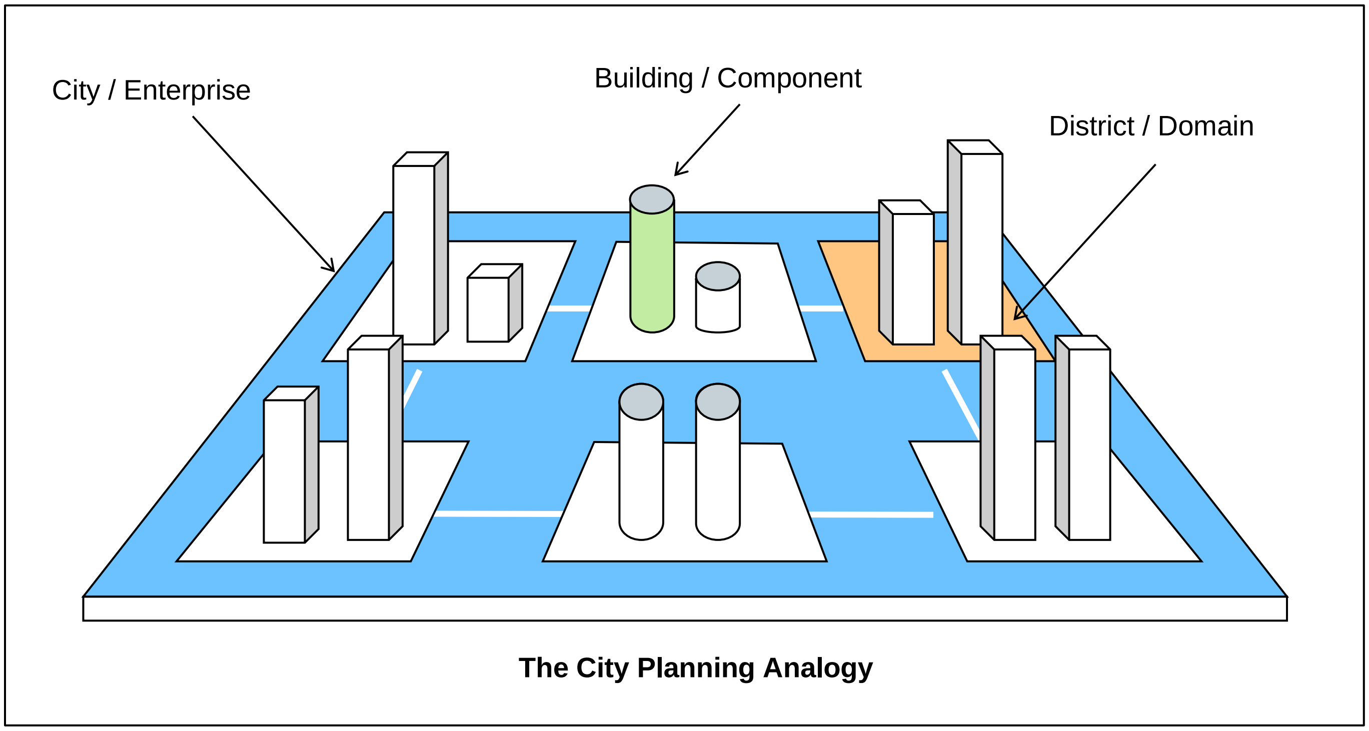

Diagram 3. City Planning

While Agile/Functional does recognise that there is a need for an emerging architecture and technical excellence it does not usually result in serious detailed consideration of the underlying enterprise, segment and capability architecture and the shared design properties that will be needed across all of the more isolated independent self-organising development teams.

5.3 The Right Pace

The concept of fast focused change underlying Agile is powerful but not singular. There are many paces not just one pace, often many dependent elements need to be put into place to enable even a simplest of functions in an effective manner. In the athletics world there a four main paces:

- Sprint
- Middle distance
- Long distance
- Marathon

Each of these paces have their place depending on the situation encountered. Using the wrong pace is not just less than good it often means a total failure. Within any one project high levels of risk may require slower paces of change to ensure that each element has properly addressed the relevant risk level; while lower levels of risk may enable a solution to be implemented more quickly. Within any change programme different iterations will often need to be addressed at different paces. TOGAF expects that the relevant iteration will be identified as having an appropriate pace and then be organised to deliver at that pace.

Agile techniques tend to promote only one pace, the sprint. However, this does not apply in large complex situations that are strongly coupled, stateful, pessimistic and controlled in nature (such as Nuclear Power Stations, Aircraft Control Systems, or non-repudiation situations) In this situation many teams may have to be involved and require significant levels of control and direction. There is a reason why the most important element of a successful large-scale military organisation is Logistics (organisation, consistency and delivery at scale - potentially in a hub and spoke architecture as in efficient goods delivery) not Commando Units (which do have their place but not at scale).

5.4 More Than Software in the Change Programme

While recent work is progressing in the Agile community on business as well as software agility it can be seen from the foundations of Agile shown above, that the focus is very firmly on software and the development and deployment of that software utilising concepts identified in DevOps.

However, most business and information system change is not the creation of new software. Rather it is often the integration of existing components (People, Organisation, Process, Information, Technology (including but not only software and computing hardware) into a workable end to end solutions in which business service levels delivered to business users are enabled by integrated building blocks; only some of which are new software.

Applying software development management concepts to the much wider arena of complete end to end business solutions using many different types of building blocks is rather problematic and should not be taken as a foundation. It depends on the enterprise that is involved in the change and the nature of that change.

Enterprises that are creating new software for themselves or clients, or to implement in either pre-package applications or connectable cloud services, may well be focused on developing and delivering new software, in which case the Agile approach to software development and deployment may be appropriate for smaller scale building blocks.

However, this is not the case for most enterprises, nor for most of the work done within most enterprises. Increasing use of and integration between existing cloud services (many of which are already running and just require implementation of API connections) and or pre-written COTS packages means that the design and build of specific new software is reducing in most enterprises and being replaced by the integration of multiple building blocks from multiple suppliers.

Often the major building blocks in information systems change projects are the business processes, roles and skills, data management and control elements, operational processes for both the wider business and information systems, providing and managing the physical or virtual locations that need to be created and utilised, training, and the development needed to support all stakeholders involved in the business service change enabled by the information systems change. While Agile as a very general set of concepts that can be applied to all of these different types of building blocks, Agile tailored specifically for software building blocks may well not be the best approach for all types of building block.

5.5 Simplicity Vs Complexity

The drive for simplicity is a constant thread in both the wider business environment and the information systems environment. However, care needs to be taken not to over simply or to avoid real complexity. As Albert Einstein is often quotes as saying "Make everything as simple as possible, but no simpler."

Unfortunately, many enterprises that make decisions in high complexity contexts can easily fall victim to excessive simplification. Agile probably overplays the small scale and simplicity concepts and in its preference for emergent rather than planned architecture under plays the need to create the underlying building blocks needed to manage complexity.

5.6 Planned vs Emergent Architecture

Emergent architectures are by definition developed and progressed on how they play out in practice rather than having clear starting positions that may enable such emergence. Planning exists for a reason. Blind emergence creates continuous failure as well as a smaller level of continuous success. The main outcome of blind evolution (evolution without a plan) is 99%+ failure and occasional success. Evolution only works when massive time periods allow the small numbers of successes to compete and improve through gradual change. Bad architecture can cause buildings and systems to fail and crash and deliver very poor outcomes despite the best of intentions.

In reality we always have some starting point for an architecture and some idea of the direction in which an enterprise would like the architecture to evolve, in line with the goals and objectives of that enterprise. The wrong starting point can be disastrous.

For example, when planning a new building the critical architectural starting point is the foundations. A building that only needs one or two floors may only need 3-6 feet of foundations. A building the needs 40+ floors may need some 60 feet of foundations. A building that needs flexibility to start smaller and grow over time may need deeper foundations for its potential future than are required for its early development iterations. The type of foundations and the materials used will also be dependent upon these considerations. Getting this staring point wrong has extensive cost, complexity and adaptability implications.

In the early stages of development of a business or information system a set of well-considered architectural choices must be made, often prior to the ability to perceive how they will emerge and play out in practice.

5.7 Who Decides on Delivery Lifecycles?

Delivery cycles need to be planned to support the development and deployment of each specific building block and their integration. However, the complete delivery lifecycle is that of deployment and use into the complete end to end business service. This deployment must be planned to support the change cycles of the enterprise and its ability perform the business transition. The ability to deliver working software into a well-managed operational environment is important but it should not be the only perspective drives a change project.

While Agile concepts from the software world do have an influence on the wider understanding of business change, there are a myriad more considerations that must be taken into account, and shoe horning an approach developed for the very narrow world of software development (within which it may also not be the best approach in many circumstances) is always somewhat questionable. The idea of frequent iterations, checking on the changing environment and requirements and feedback from what works in practice, is not just local to Agile concepts but has been an integral part of all change practices for many decades. This does not mean that all enterprises manage change well, and some manage change very poorly, but broader well managed change should be addressed from broader perspectives and considerations than just Agile approaches.

5.8 Shaping the Overall Change Programme

A key consideration at the beginning of any change programme is how to structure and approach the change. In the 1990s this issue was extensively researched as the information systems industry looked to learn from the benefits and problems encountered when taking waterfall or prototyping approaches to change. The outcome of this research was an integration of most of the previous methods into a generally agreed and unified approach and framework for change which is now used under the terms Unified Process (UP) with an associated architecture and design language Unified Modelling Language (UML). At the core of this approach is a series of characteristic phases representing the type of work done in major change:

- Inception - prepares an organisation to agree upon and address a change with some delivery to test reality.
- Elaboration - address the change at a relevant level including early implementation of key deliverables.
- Construction - applies the architecture to create and increasing implement the change.
- Transition - complete the implementation and governance of the change.

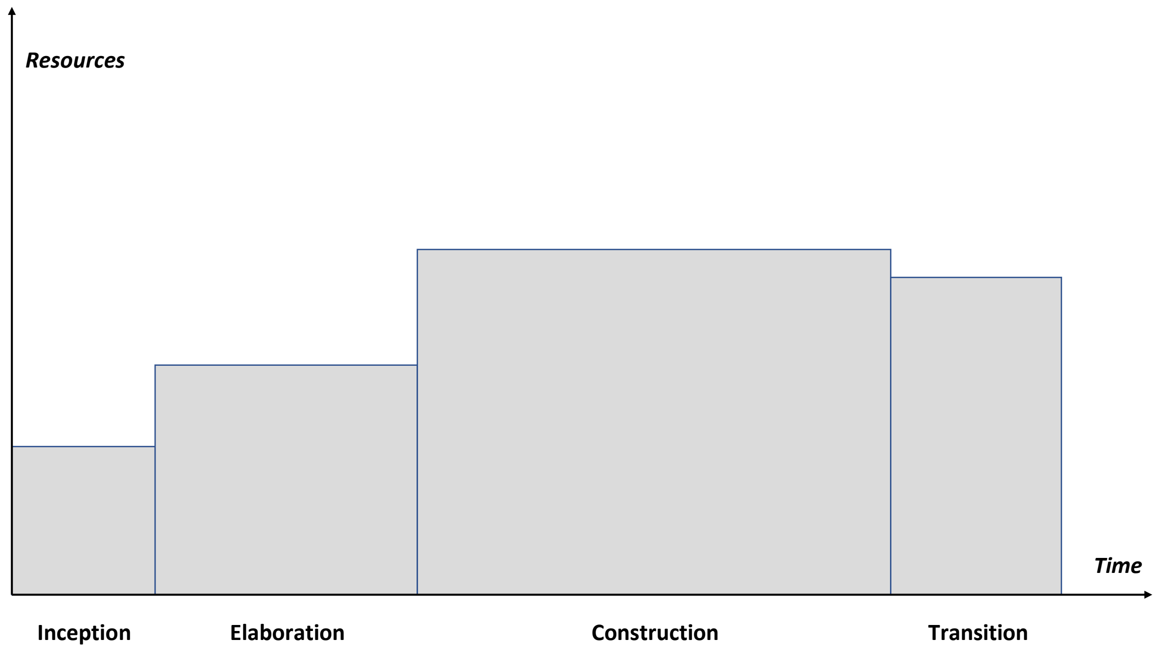

Diagram 2: The Unified Process Delivery Phases

These phases are organised into a series of time boxed, value and risk driven iterations of work that start to deliver real implementations as early as possible in major change but also continually respond to, and shape changes in, the enterprise architecture within which the change is being implemented.

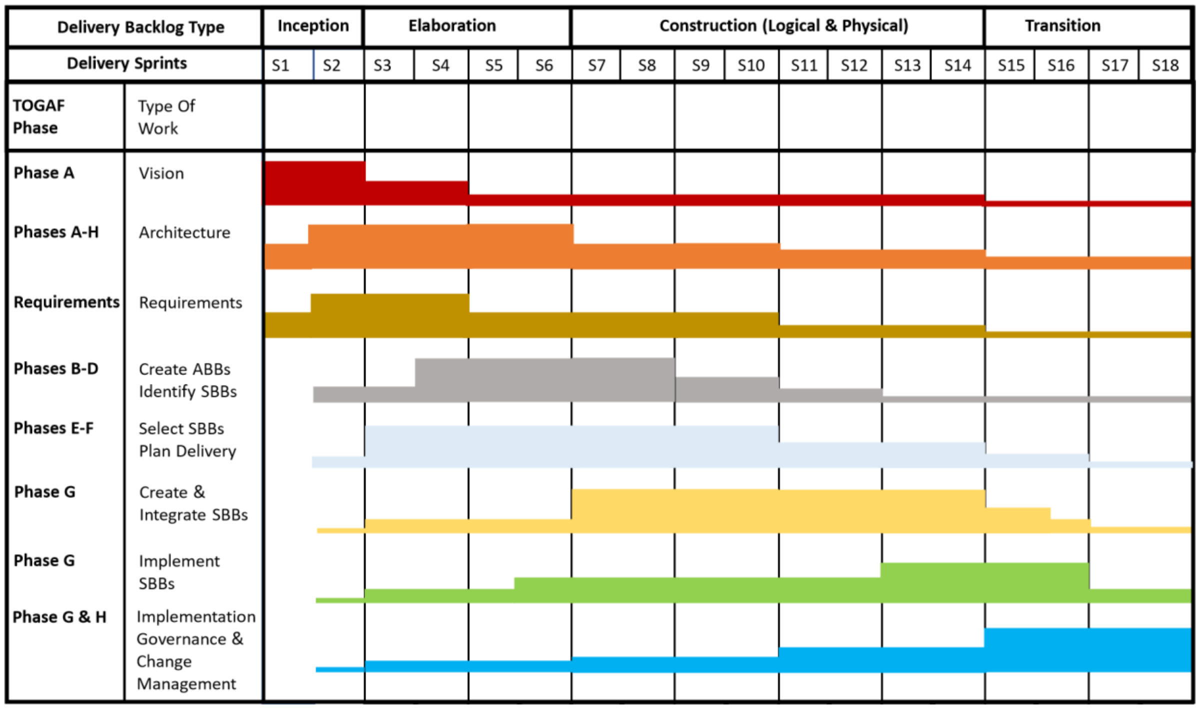

Diagram 3: Implementing change incorporating TOGAF, Agile and UM Delivery Phases

Diagram 2 shows how TOGAF, Agile and the Unified Method delivery phases can be implemented in an effective manner that directly maps to the way the Agile change is implemented.

In the Agile approach an initial workshop (or longer period of work if needed) of some form is run to identify the nature of the change, mobilise the stakeholders and other resources and shape the change into a backlog of potential packages of work (sprints). This is the equivalent of the Vision phase in TOGAF and prepares an Agile change programme for action. This initial workshop needs to highlight the requirements, often structured into user stories and acceptance criteria, and start to shape an architecture and integration approach against which each sprint can deliver outcomes that can be combined into a coordinated solution as they are complemented.

Once the backlog and appropriate partitioning of change into sprints has been done each sprint can be started, utilising a team of as small as possible, but relevant scale, to progress that sprint through from Vision to Governed implementation.

- Earlier sprints will do more of the early type of work (understanding and shaping) but will also deliver some working solutions (using all of the needed building blocks to deliver the business service not just the software).

- Later sprints will do more of the later type of work (implementing and testing) but will also have to address any emerging changes in requirements, vision and architecture.

- As each sprint is completed the learning from that sprint is evaluated and the backlog refactored to the degree necessary to respond to that learning and any subsequent changes to requirements. This refactoring of the backlog feeds up into any changes needed to the Vision and enterprise architecture.

In this way once an early understanding is generated on the change solution vision and the architecture that will enable that vision to be achieved effectively; change that uses an Agile style delivery cycle can easily be implemented within the wider TOGAF framework for managed change.

5.9 User Story Driven Design for Simple Restricted Contexts

One of the limitations of the traditional Agile approach is that the surrounding structure for change is usually that of user stories and acceptance criteria, and then testing and governance based on meeting those acceptance criteria. This essentially describes the change from the user/customer perspective and aligns with a common focus in Agile on customer journeys rather than complete end to end processes and relevant internal detail.

While this provides a user's perspective of the change, it provides nothing to suggest how the change can be delivered, what good building blocks would be, nor how they should be integrated to deliver the business service levels that the change is there to deliver. It is externally focused around the requests made on the systems and subsystems needed but not internally focused on how those systems and subsystems should be built and integrated, nor how they will consequently be able to evolve over time.

In small scale simple change this Agile approach can be very successful and can provide enough direction to small teams to develop and deploy relatively simple components with restricted integration in fast cycles of change.

However in larger scale more complex change there is a need to understand in more detail the specific nature and internals of the building blocks and components; such that they can be well made and deliver their non-functional (i.e. performance and quality aspects) properties, including evolving over time.

Additionally in order to approach the implementation of the user stories with technical excellence in line with Agile Alliance principle no. 9 - "Continuous attention to technical excellence and good design enhances agility" there is a need to look deeper and understand the basis for such technical excellence when delivering flexible, component based systems and subsystems; not just understand the externalised user story and customer journey perspective.

5.10 Extending the Context Addressed by User Story Driven Design with Additional User Stories

Many organisations have found that one type of user story (focused on the persona's external perspective) does not provide enough information from which a good solution can be implemented. As a result, additional story types are often created to uncover the increasing level of requirements and detail needed. Examples of these in order of level of detail are:

- Customer Story - the traditional user story from the external persona's perspective.

- Demand Story - the requirements that an enterprise implementing a customer story needs to add to make it work in their specific situation (particularly when constrained by rules, regulations and limitations on solution building blocks and components)
- Component Story - the requirements (including constraints) that a development and delivery organisation needs to be able to understand to define and integrate building blocks and components within their specific architecture.
- Engineering Story - the requirements (including constraints) that a build team needs to be able to understand to specifically engineer a building block or component using specific elements and technologies.

The depth of story required usually emerges recursively as a backlog is progressed based on the information required to make a success of each sprint. Generally, the early sprints in the backlog would uncover the levels of detail required and the later sprints would move to work at those levels of detail. This extension ensures that the requirements are understood at an appropriate level of detail but still does not provide a basis for the architecture and technical excellence of the systems, sub-systems, building blocks and components that need to be delivered.

5.11 Further Extending the Context from User Stories to Use Case Driven Design

For Agile teams wishing to address larger and more complex change; use case driven design provides the missing element from the Agile approach. That of a coherent approach to more a detailed understanding of the requirements, the nature of the building blocks and components needed to implement those requirements, how they can be integrated, and also the internal structuring of the building blocks and components being implemented to deliver the change.

Where specific approaches are needed for defining the building blocks and components to deliver a change, TOGAF recommends the use of the UML (Unified Modelling Language) or its own UML like full architecture description language Archimate. In both of these event and activity modelling using use cases and/or process models are a core part of the method addressing the essence of any change.

Use Case driven design was developed by Ivar Jacobsen for Ericsson in the 1980s to enable the creation of re-usable blocks of functionality that responded to the requests made by external actors. The nature of the building blocks and the change to develop those building blocks would be based on each external actors' concept of value. Essentially a use case is: "A goal directed sequence of steps that delivers value to the user of that case". Use Case driven design was one of the original techniques included in the UML.

The Use Case approach starts with a diagram showing actors interacting with a system. Each interaction points to a use case inside the system that at its simplest provides a summary post-condition statement of what an actor wants to happen within the system of concern and share with any other actors. In the original approach "Objectory (Design with Building Blocks)", and the subsequent book "Object Oriented Software Engineering A Use Case Approach 1992" that post-condition was a paragraph of text describing what the actor wanted. This is identical to the current concept of a user story.

Mike Cohn in "User Stories Applied" 2004 (one of the bibles of the user story approach) comments at length on the difference between user stories and use cases. He says that a user story is shorter, simpler and a basis for planning, that may not need to be kept of the full solution/product lifecycle; while a use case is longer, more complex, a basis for implementation as well as planning, and would normally be used across the whole solution/product lifecycle, providing key documentation to understand how and why it is the way that it is. However, Cohn misses the key overlap between the two approaches, that the post condition of a use case is essentially the same as a user story. This type of thinking has driven a wedge between the two approaches that does not reality exist. It is merely a question of the level of detail required to achieve the most appropriate solution.

User stories provide for this same externally focused high level of description but identify actors as external agents with characteristic personas. In the use case approach a statement on the performance and quality attributes that describe how the use case should perform and be tested is then added. This is similar to the acceptance criteria paired with user stories in the Agile approach. The beginnings of the two approaches to understanding requirements and identify the requests being made on the systems and subsystems needed to meet those requirements, is therefore almost identical in both user story and use case driven design.

For larger and more complex situations needed the use case post-condition is supplemented by a pre-condition statement and identification of the steps seen by the actor within each use case, a definition of alternate courses that may be needed, and a statement about any exceptions paths that may be required. These use case descriptions provide a complete definition of the external event (rather than activity) requirements on a system. In addition, a set of interface, controller and entity objects are defined that provide the basis for implementing those requirements in an information or human system

Where the change can then be addressed and a solution delivered by skilled developers working directly from this more detailed statement of requirements and the base building block set, then further detailed design would not be needed before the build and integration of the required building blocks and components (reused, bought in or built).

Where a change is of greater, scale, complexity or risk, the use case driven design approach was extended in the UM. This incorporated additional elements to understand the systems behaviour: through event analysis and design (in interaction and timing diagrams), activity analysis and design (in process activity models), state transition analysis and design (in state transition diagrams), object definition and behaviour design (in sequence and communication diagrams).

It also addressed the systems structure through: entity/class analysis and design (in class and object diagrams), groupings of any set of systems, sub-systems, building blocks or components and their interactions (in package diagrams), components and their interactions in (component diagrams), hardware and its relationship to software components (in deployment diagrams) and locations and their deployed hardware and software components (in locality diagrams).

While this may seem to be a lot of elements and steps; they are only used when there is a need to work at that level of detail and scope. Too many unnecessary steps and the pace of change is slowed down, while too few unnecessary steps and the change being delivered becomes partial, unreliable or unusable.

TOGAF describes change in very similar terms to use case driven design. It contains an approach based on the creation of business capabilities to enable the delivery of business services, implemented in physical and software (or other nonIT), building blocks made up of one or more artefacts (components).

The role of the architect is to understand these elements and possibilities and make good decisions on how best an enterprise can deliver each specific change and progress general change, in line with its strategy and in this way deliver the right level of technical excellence together with the most appropriate pace of change.

5.12 Integrating Many Lifecycle Approaches Using SAFe

SAFe is an approach that looks to integrate Agile delivery lifecycles with Enterprise scale change. It grew out of an excellent book “Scaling Software Agility” by Dean Leffingwell in 2007. Its key elements are:

- Core Values - SAFe’s core values describe the culture that leadership needs to foster and how people should behave within that culture in order to effectively use the framework.
- Alignment - SAFe requires that companies put planning and reflection cadences in place at all levels of the organization. With these in place, everyone understands the current state of the business, the goals, and how everyone should move together to achieve those goals. By synchronizing people and activities regularly, all levels of the portfolio stay in alignment. Information flows both upward and downward in a timely fashion, unlike traditional top-down, command and control structures.
- Built-in quality - In the SAFe framework, agility should never come at the cost of quality. SAFe requires teams at all levels to define what “done” means for each task or project and to bake quality development practices into every working agreement. According to SAFe, there are five key dimensions of built-in quality: flow, architecture and design quality, code quality, system quality, and release quality.

SAFe does not replace TOGAF or Agile but assists organisations working within the TOGAF framework to choose the most appropriate lifecycle delivery approaches.