Title	Agile enterprise system design using configurable Software-as-a-Service
Sub Title	
Author	Lin, Zhihong John(Haruyama, Shin'ichirō) 春山, 真一郎
Publisher	慶應義塾大学大学院システムデザイン・マネジメント研究科
Publication year	2021
Jtitle	
JaLC DOI	
Abstract	
Notes	修士学位論文. 2021年度システムエンジニアリング学 第336号
Genre	Thesis or Dissertation
URL	https://koara.lib.keio.ac.jp/xoonips/modules/xoonips/detail.php?koara_id=KO40002001-00002021-0010

Master's Dissertation 2021

Agile Enterprise System Design Using Configurable Software-as-a-Service

Lin, Zhihong John
(Student ID Number : 81934607)

Supervisor: Haruyama, Shinichiro

September 2021

Graduate School of System Design and Management,
Keio University
Major in System Design and Management

SUMMARY OF MASTER'S DISSERTATION

Student Identification Number	81934607	Name	Lin, Zhihong John

Title

Agile Enterprise System Design Using Configurable Software-as-a-Service

Abstract

From a system engineering perspective, an enterprise is a complex system. Designing an enterprise system is acknowledged to be an ill-defined and complex problem. However, the practice of enterprise systems architecting is often plan-driven and involves a lot of time-consuming and costly upfront work. This approach does not cope well with the nature of enterprise design, where there is often no definitive problem statement and no "stopping rule" that signals the end of the design process. Furthermore, this kind of "Big Design Upfront" (BDUF) quickly becomes outdated and irrelevant as the enterprise and its associated requirements may change fairly quickly over time. In contrast, agile software development methods are designed to handle this kind of complexity and changing requirements. In particular, emerging agile enterprise architecture strategies and tactics are a way to incorporate enterprise architecture considerations into the software development process when needed and to the extent that it is useful in solving a problem. This paper attempts to validate the hypothesis that agile enterprise system design produces a system with better quality attributes compared to a plan-driven approach. Through action research, a three-month-long experiment is conducted on a live enterprise project to implement this agile enterprise design approach. The experience of the team members and the output of this approach is compared with their original approach in order to determine whether there was any improvement. Improvement is measured in six dimensions that represent the quality attributes of the enterprise architecture. For this project, the agile enterprise design approach was determined to be better than the original plan-driven approach in the product usability, process efficiency, implementation speed, maintainability and cost dimensions.

Key Words

Enterprise Architecture, Agile, System Architecture, Software Architecture, System Design

I. INTRODUCTION

This paper articulates how to conduct enterprise system design in an agile manner. It then attempts to answer the question of whether doing so produces a better output than the traditional plan-driven enterprise system design approach. Enterprise system design (or enterprise engineering) is a comprehensive approach to designing a system, including aspects such as organisational design, information design, and business process design in addition to technology design, so as to achieve optimal system outcomes. However, the common criticism is that it is not a practical approach because there is too much upfront planning attempted for a context that is very complex. Agile on the other hand is an approach from software development that is good at handling complexity, and delivering a small, usable or working system, and building out in increments. The main challenge with agile is in scaling that approach to cover an extensive scope like that of typical enterprise system design problems. To date, there are no known research papers on enterprise system design done in a truly agile manner (although there are system design approaches like Bondar et al. [1] for system-of-systems that are iterative). There are also very few research papers with evidence on successful scaled agile projects – the most notable one being Paasivaara et al. [2]. The novelty of this research is therefore to demonstrate how agile enterprise system design has been conducted successfully in an actual project, with a better output than the plan-driven approach.

The basic structure of this paper is in three sections. Firstly, definition and clarification of key concepts. This section also reviews the existing research literature. Secondly, a description of the action research method. This details the approach and documents the results. Third, observations, analysis and learnings. The objective of this paper is not to propose a new framework or methodology, as there are already many agile and hybrid approaches in practice. Rather, the intent is to illustrate what enterprise system design might look like using an agile rather than plan-driven approach.

II. LITERATURE REVIEW OF KEY CONCEPTS

A. Enterprise System Design

When “enterprise system” is used in this paper, it is used in the system engineering or system design sense to refer to the enterprise as a complex system, and not an enterprise software system like an Enterprise Resource Planning (ERP) or Customer Relationship Management (CRM) system. The difference is that the former has more extensive scope, treating the enterprise as a system-of-interest (SOI) that includes aspects like organisational structure, business process, information structure and flow, on top of technology. There are many frameworks developed to cover these different aspects of the enterprise system, with Zachman [3] and The Open Group Architectural Framework (TOGAF) [4] among the more widely recognised, but also Nightingale and Rhodes [5], Rouse [6], Giachetti [7], etc. The TOGAF standard considers an "enterprise" to be any collection of organisations that have common goals. This includes a whole corporation or a division of a corporation, a government agency or a single government department, geographically-distant organisations linked by common ownership, or partnerships and alliances of businesses working together [4]. The common theme across these authors is that “enterprises must be designed in a holistic manner that is informed by systems thinking […] and the advocacy that the IT dimension of an enterprise is one of many that must be designed and integrated with the others” [3]. This is essentially a systems engineering approach, with the SOI as the entire enterprise [8].

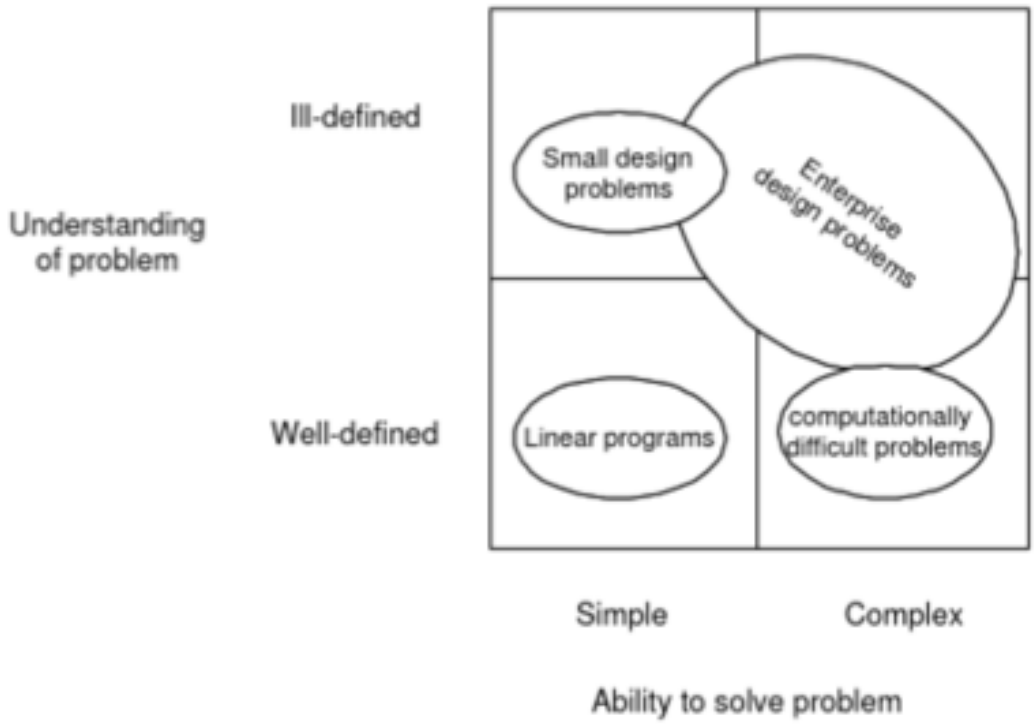

Fig. 1: Enterprise Design Problem Typology [7].

Designing an enterprise system is an ill-defined and complex problem. This is illustrated in figure 1 above, reproduced from Giachetti [7]. As Giachetti argues, clarity of scope is a significant issue. There is often no definitive statement of the problem, and each stakeholder may have a different perspective depending on their experience and role, and all can be right at the same time, depending on their viewpoint. They may also be no definitive solution, and therefore no "stopping rule," so the design process could go on indefinitely. To understand the problem, the designer creates a solution and tries to gain greater knowledge about the problem in order to generate another solution. There may be many competing solutions that are neither correct nor wrong, only good or bad, in relation to the design objectives. [7]

To describe the "design" of an enterprise system, we rely on the concept of architecture. The architecture of the enterprise system is the essence of the system. Since the enterprise system is very complex, it is important to determine key aspects (and not try to detail the entire design) in the architecture. This is echoed in various definitions of architecture – e.g. "software architecture is the decisions which are both important and hard to change", or "the decisions you wish you could get right early in the project" [9]. The system exhibits an architecture, and this architecture is usually expressed in the form of an architectural description or physical documentation, such as specific architectural views, or an explanation of architectural rationale [10]. In fact, most of the frameworks cited above are called "enterprise architecture" frameworks because they describe how to come up with an architectural description that is an aggregation of the many architectural views. Each architectural view addresses a particular aspect or concern such as the business process, or the flow of information etc. In this paper, we focus on the practice of "architecting" – i.e. the process that the architect does – rather than architectural descriptions [11]. The output of the process of architecting is an architectural prototype.

The final note of importance regarding enterprise system architecting is that the literature is extensive, but the approaches are largely plan-driven, and at most iterative or incremental, not agile. The evolution of enterprise architecture (EA) frameworks is covered in Gong and Marijn [12], reproduced in Figure 2 below.

EA is a new discipline about 30-40 years old. It arose from management engineering and information systems disciplines, with early beginnings in the Business Systems Planning (BSP)

methodology by IBM (1975) and the PRISM Enterprise Architecture framework (1986). The Zachman framework (1987) is most commonly referenced as the start of EA, with the first EA methodology introduced by Spewak and Hill (1992) in Enterprise Architecture Planning (EAP). Today, The Open Group Architecture Framework (TOGAF) standard is arguably the most established and well-known EA framework, with over 100,000 certified architects globally. The range of Gong and Marijn's chart is only until June 2017, and does not include the most recent update to TOGAF – version 9.2, released in April 2018.

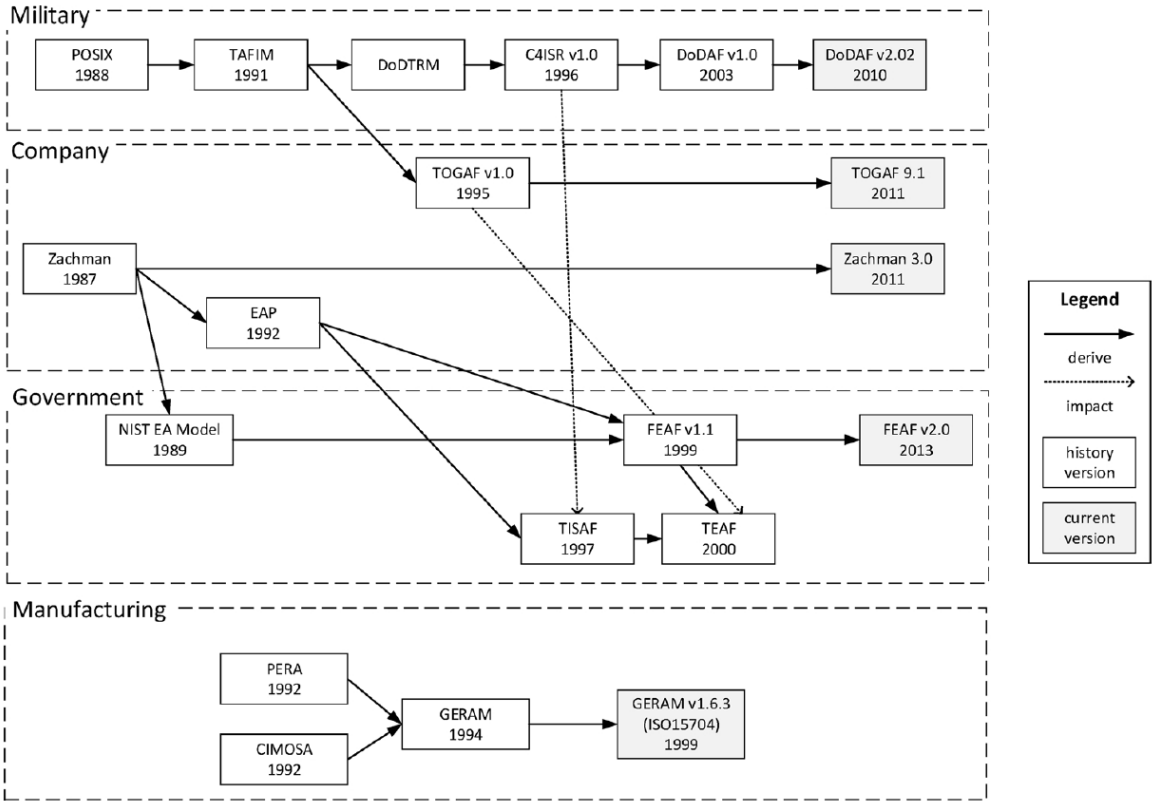

Fig. 2: The development of EA frameworks in different domains (to June 2017) [12].

The first version of TOGAF was based on the Technical Architecture Framework for Information Management (TAFIM), developed by the US Department of Defence (DoD). Given its defence origins in designing and managing large complex system, there is a lot of big upfront planning to cover the many different aspects of the enterprise system. Each aspect may also require different views, leading to very extensive documentation. This can be very time consuming and it is also difficult to keep it up to date, as the enterprise is continually changing. Also, businesses do not operate in this manner in reality – there is planning and early design, but typically it is a process of trial and error in responding to the external environment. This

may explain the lack of real-world use of such enterprise system design approaches. Unlike software development, there was no "agile revolution" in enterprise architecture, although frameworks like TOGAF now emphasise extensive iteration, as seen in figure 3 below [13].

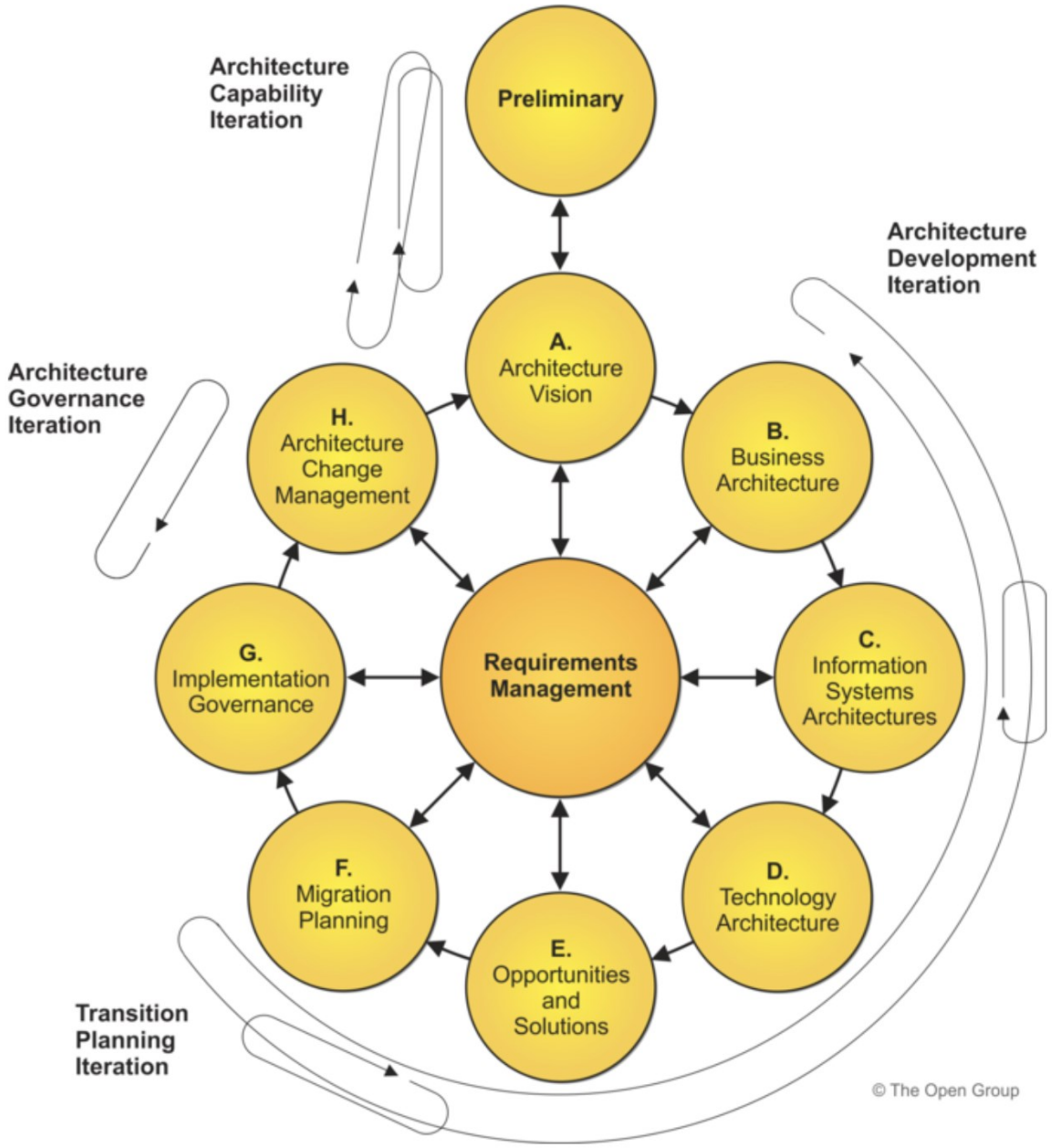

Fig. 3: Iteration in the TOGAF Architecture Development Method (ADM) [13].

B. The Agile Approach

Agile software development is an approach developed to handle complexity and changing requirements. Enterprise software systems are often developed in a highly complicated

environment and this complexity is both within the development environment and the target environment. There are many "flavours" or types of agile software development, but the common priority is to ensure early and frequent delivery of working software that the customer finds valuable [14]. This allows assumptions to be tested and refined based on actual user behaviour in response to the product.

Agile software development approaches are empirical ("black box") system development methods. Instead of pre-defined processes, they use controls to manage unpredictability and control risk [15]. As the complexity of the project increases, the greater the need for controls, particularly ongoing assessment and response to risk. Scrum is an example of an agile software development approach. The Scrum approach assumes that the analysis, design, and development processes in the Sprint phase are unpredictable. As such, instead of linear or pre-defined (possibly iterative) processes, Scrum has "events" or "ceremonies" such as Sprint Planning, Daily Scrum, Sprint Review and Sprint Retrospective, to respond to the changing external environment.

Agile software development has been very successful in small teams and organisations with standalone software projects. More recent developments have seen the introduction of scaled agile frameworks such as Scaled Agile Framework (SAFe), Large Scale Scrum (LeSS), or Disciplined Agile Delivery (DAD) being introduced with some success [2]. The main reason for the introduction of these larger frameworks is the need to take into account the broader enterprise system (here used in the enterprise engineering sense) when building enterprise software systems. There is a need to interface with and integrate with existing processes and organisational structures [16]. Because of their empirical nature, Agile methods provide little guidance on how agile teams should interact with the environment at large. Some agile practitioners even completely eschew the more plan-driven enterprise architecture approaches such as TOGAF (since even software architecture decisions are deferred, what more enterprise architecture). Scaled agile frameworks recognise that enterprise architecture is still important, but try to introduce the practice in a more agile way. More reconciliatory proponents call for "agile enterprise architecture", that can be practiced in varying ways depending on the needs of the context [17].

While Agile has its roots in software development projects, it is more widely recognised today as a valid project management approach. The Project Management Institute (PMI) identifies

four project life cycle approaches that distinguish Agile from Predictive, Iterative, and Incremental approaches. The main characteristics of agile project management are dynamic requirements, activities that are repeated until correct, frequent small deliveries and customer value via frequent small deliveries and feedback. It is this agile project management approach that is being applied to enterprise system design in this paper. For the rest of this paper, when we compare agile to other "plan-driven" approaches, "plan-driven" will refer to any combination of the other three project approaches in Table 1.

	Characteristics			
Approach	**Requirements**	**Activities**	**Delivery**	**Goal**
Predictive	Fixed	Performed once for the entire project	Single delivery	Manage cost
Iterative	Dynamic	Repeated until correct	Single delivery	Correctness of solution
Incremental	Dynamic	Performed once for a given increment	Frequent smaller deliveries	Speed
Agile	Dynamic	Repeated until correct	Frequent small deliveries	Customer value via frequent deliveries and feedback

Tab. 1: Characteristics of Four Categories of Project Lifecycles (from Agile Practice Guide)

C. Software-as-a-Service (SaaS)

In the enterprise engineering literature, not enough attention is placed on software (and broader technology) constraints on the enterprise system. It is often assumed that software is flexible enough to do most of what is required of the business. However, in practice, the technology may not be mature enough to meet a specific business process requirement, or even if it is able to do so, it may come at the cost of over-engineering – too much time spend developing, or a software system that is costly and inefficient to maintain. With Software-as-a-Service (SaaS) products in particular, the software is not completely flexible and there needs to be a realistic expectation of what the technology can do. This is because SaaS products are designed for speedy implementation, with an existing design to support established business best practices. SaaS has also evolved to a point of maturity where it is highly configurable, much like Commercial-Off-The-Shelf (COTS) software solutions, the difference being SaaS software is

hosted in the cloud, and not on-premise like with COTS software. In this sense, SaaS deployment is even faster, as the infrastructure to support the software is already in place and does not need to be installed.

When it comes to using SaaS software, it may actually be better to fit the business to the software than to try and fit the software to the business. It is certainly possible to customise SaaS to a large extent, where the extreme case would be to just use the SaaS system as a database with a completely separate user interface layer, but that would defeat the benefits of buying a SaaS product in terms of speed of deployment and cost. Modern SaaS also has the benefit of regular software updates or patches that improve the capability of the software. If the system is customised too heavily, the frequent updates might result in a higher maintenance cost to ensure that existing customisations do not conflict with new features being released.

A good enterprise system architecture should take into account the constraints of SaaS software systems, as architecture is about good fit between form to be designed and its context [18]. Many fit-gap assessments for SaaS system take the SaaS system as the system-of-interest and the business process as the external context, where technical requirements are translated from external business requirements. The SaaS system is then customised to meet those business requirements. Better approaches try to change the business process (and their resulting requirements) to reduce the extent of SaaS customisation needed. However, few projects take a holistic enterprise design perspective where the organisational structure, business process and information structure and flow, together with the SaaS system are all part of the system of interest being designed. This last approach is what is being attempted in this paper.

There are many benefits of using SaaS systems, both for large and small enterprises. These include productivity improvement, quality improvement, customer service improvement and cost and inventory reduction, depending on the type of SaaS system being considered [19]. However, the purpose of this paper is not to justify the general benefits of using SaaS. The purpose of using SaaS in Agile Enterprise System Design is two-fold. Firstly, because SaaS is fast to configure and implement, and allows us to accelerate the technology development component of system design to see results quickly. This in turn allows us to see results from the action research study (outlined in section “II. Method”) within a timeframe of 3 months. Secondly, SaaS is semi-rigid in terms of configurability, so when adopting an enterprise system

design approach, it forces the project team to consider changing other components of the holistic system in order to achieve a good fit.

D. Agile Enterprise System Design (ESD)

In the traditional approach, the activity of architecting is done first to provide a high-level design of the entire enterprise that will guide all other enterprise projects. An architecture represents significant, broad design decisions for the enterprise, from which all other design decisions should be consistent. Architecturally significant decisions are those that the architect (or architecture team) needs to make in order to address the concerns of strategy, structure, and enterprise integration. Architectural decisions include deciding how to decompose the enterprise into views, how to decompose each view into different abstraction levels, policies on technology usage, decisions on business culture to guide organizational design, and decisions on what modeling conventions to use [7]. Conway's "Law" argues that the design of a system tends to be biased by an organisation's communication structure [20]. Without EA, a software system is at risk of simply mirroring the structure of the organisation that designs it.

The agile movement has long been critical of developing EA as a separate activity. Their argument is that this kind of Big Design Up Front (BDUF) is easily outdated as requirements often change over time. It is also time-consuming and costly, and stakeholders are not able to see its value until much later, if at all. However, this does not mean that architecture is not important. In fact, there are architecture functions in some of the scaled agile frameworks like SAFe, and defining the "bounded context" or system context is critical in the microservices movement. Instead, Agile proponents argue that architecture is in essence a shared understanding. In most successful software projects, the expert developers working on that project have a shared understanding of the system design. This shared understanding is a social construct because it doesn't just depend on the software, but on what part of the software is considered important by group consensus [9].

The main difference in the application of "agile" architecture then is in the concepts of "just-in-time" and "just-enough". Just-in-time refers to deferring or leaving architectural decisions until the last possible moment instead of anticipating or planning in advance [21]. This is the preferred approach in Extreme Programming (XP) (where code can always be refactored) and also the basis of "emergent design", where an architecture emerges over time. Just-enough

refers to the amount of upfront planning required, depending on the project complexity. Using data from 161 industrial projects, Boehm et. al. attempt to show the "sweet spots" for the amount of architecture that should be conducted up front (Fig. 4.). Devoting much, if any, time to up-front work is a waste for a small project. But for enormously complex projects, it is difficult to imagine how Agile principles alone can cope with this complexity if there is no architecture to guide and organize the effort [22]

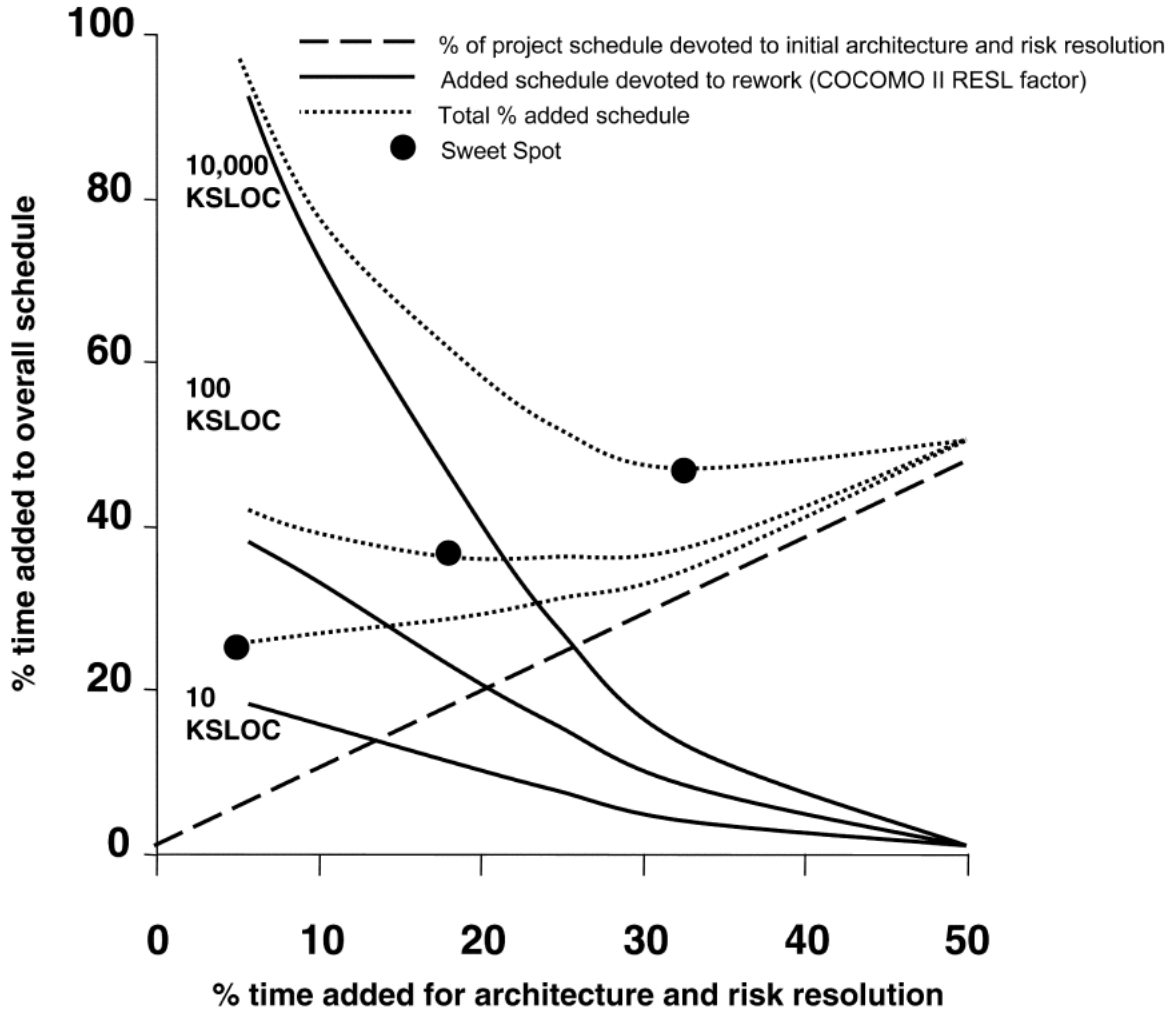

Fig. 4: Sweet Spots with just enough Architecture [22]

E. Research Novelty

The novelty value of this research is in proposing and validating agile ESD, which is a new approach spanning two major research domains. From the enterprise engineering or enterprise system design research landscape, agile ESD is novel because there are no known papers that

attempt a comprehensive ESD scope while using an agile approach. For example, established authors like Martin (2008) [8], Giachetti (2010) [7], Nightingale & Rhodes (2015) [5], and Lapalme et al (2016) [3] all use plan-driven approaches for complex enterprise transformation. This is because plan-driven approaches can be described in terms of lifecycle stages and process steps, and this is helpful when breaking down complex projects with enterprise-wide scope. Only recently have papers in the ESD domain, such as Bondar et al (2020) [1], started to use the term "agile", but they actually mean iterative (refer to table 1), which is still considered a plan-driven approach. In fact, if we consider how industry is usually slightly ahead of academia in terms of adopting new practices, even TOGAF only recently introduced iteration (still a plan-driven approach) to its EA process, indicating there is still some way to go before agile can be assimilated into mainstream ESD. The challenge with advocating an agile ESD approach is that it is difficult to define exact process steps, as agile is an empirical or "black box" system development method [15]. This is probably why ESD papers have avoided using such an approach. We take on this challenge in describing an actual agile approach to ESD using a single case study and action research methodology, which is covered in the next section. From the agile software development domain, which is the home of agile practices, there are no papers where agile is applied to an enterprise system as defined in section IA. Section IIIF will go into more detail about how agile ESD is different from typical applications of agile to software or hardware system design, but the main thing to note is that ESD includes people in the SOI, whereas typical agile projects described in the research literature have people as part of the context, outside the SOI. For a paper like Dingsøyr (2006) [23], which is closest to this paper in terms of using action research methodology on a single project, the SOI is a software system. Agile is known to be effective in small teams, so the challenge that is of recent research interest is about applying agile to large, complex projects. Papers in this domain focus on how to scale agile "enterprise-wide". However, even for papers like Ambler (2008) [24], Waardenburg & Vliet (2013) [25] and Paasivaara et al (2018) [2], the large-scale or enterprise systems they study are still mainly technical systems – hardware, software or system-of-systems – with people (or users) as part of the system context. This is also the case for other papers like Bass (2014) [26] and Imani & Nakano (2018) [27], who try to propose hybrid plan-driven and agile approaches in order to scale agile.

Besides proposing agile ESD, the novelty value of this paper is also in attempting to validate agile ESD as an effective or beneficial approach. This directly addresses the question of why there is a need for agile ESD in the first place (as not every innovation is useful or valuable in

and of itself). In theory, agile ESD combines benefits from both agile and ESD. From ESD, the benefit is the comprehensive design approach with views across business process, information structure & flow, organizational structure and technology. Both ESD (as a plan-driven approach) and agile have different ways of handling complex system requirements that are effective in their own way. Agile is particularly strong in providing a flexible response to unpredictability (or changing requirements) and is focused on delivering incremental customer value quickly. By combining agile and ESD in the way proposed in this paper – "agile ESD" – we should see a combination of these benefits. This is what we seek to validate through the single case study and action research project described in the next section.

III. METHOD

A. Action Research and the Single Case Study

This paper draws on the single case study [28] and action research [29] methods to demonstrate and validate the effectiveness of agile enterprise system design. We refer to the setup of this action research as "the action research project" or "the project" in short for the rest of this paper.

Action research is a method introduced as early as 1946, and aims to contribute to the both the practical concerns of people in an immediate problematic situation as well as to the goals of social science by joint collaboration. Action research is a response to the perceived deficiencies of positivist science, where knowledge is derived only from sense data that can be verified between independent observers. It is suited to our subject of an enterprise organisation as the system of interest, as "organisations are systems of human action" and "organisations can be legitimate objects of scientific inquiry only as single cases" [29]. It is also suited for research on empirical "black box" approaches like agile, without predictable process steps that may be consistent across different projects. Such a method has been applied in a similar domain by T. Dingsøyr et al. [23] to research Scrum in a small cross-organisational software project, and this paper takes reference from that.

Since each organisational system is a single case, we use the single case study method for data collection and analysis. Sources of evidence include documentation, interviews, direct observation, participant observation and physical artifacts [28].

B. Action Research and Agile Project Management

We use the structure of agile project management to support the iteration in this action research project. Action research is a cyclical process with five steps [29]. First, diagnosing: identifying or defining a problem. Second, action planning: considering course of action for solving the problem. Third, action taking: selecting a course of action. Fourth, evaluating: studying the consequences of the action. Fifth, specifying learning: identifying general findings. The entire project is run according to the stages of the action research cycle.

First, the problem is diagnosed. In this project, the identified problem involved the creation an integrated enterprise system to manage marketing and sales activities. There was an existing system, but a new system was needed to increase the scale of the sales and marketing activities without requiring a corresponding increase in personnel. This meant the adoption of new technology – specifically a new Customer Relationship Management (CRM) software system, as well as defining a new business process. As the business process spanned multiple departments, and the knowledge of how to create such a system was cross-functional, members from different departments in the organisation needed to work together to deliver this outcome. For example, the marketing team was one department that worked with many sales departments. The sales teams would be divided by geography and also by the type of customer. The marketing and sales process for each of these customer segments may have variations. While the business problem was to create an efficient system that allowed each marketing and sales person to handle many more customers (or potential customers) than before, the organisational problem was what approach was best to combine the different expertise and manage the project to ensure a successful outcome.

Second, action planning. In terms of the approach, the team entrusted with the creation of this new system had already done work in a plan-driven or waterfall project management style. This included preliminary business and technical requirements derived from identified business needs, work breakdown, and assigning members responsible for those activities. More specifically, there were four parallel threads in the work breakdown structure (WBS) broken down by key activities – design, development, system integration, and project administration – with dependencies coordinated across the activities. However, as the eventual system would require a new process, there were still many unknown and undefined requirements, which made

it difficult to continue the project using a waterfall approach. The alternative would be to switch to an agile project management approach, and some of the team members were already familiar with this approach from their past experience in other organisations, or from other projects in the same organisation. An agile approach would be good in order to develop and test the new system and get feedback from internal customers – i.e. the users of the system in the context of the new business process. However, unlike typical agile software development projects where the business requirements are exogenous to the project, for this project, the business requirements could be endogenous to the project – i.e. the stakeholders were open to consider changing the existing business process, or designing a new business process where one did not previously exist. As such, this business process design would need to be factored into the project management approach.

The project team agreed to try out an integrated agile enterprise system design approach as there was no suitable alternative known to them. Compared to the prior waterfall approach, an agile approach was preferred because there was a significant degree of requirements instability. There were business requirements that could not be defined upfront (due to the absence of an existing business process) and could only be defined after determining the effectiveness of the new process. There were also many existing business requirements that might change during the course of the project if the business process were to change. However, the typical agile software development process did not exactly meet the needs of the team as the business requirements were endogenous to the project – i.e. a factor that could be changed and defined through the course of the project, and something that could also be affected by other factors within the project. For example, a limitation or constraint in the SaaS software product that might warrant adapting the desired business process to better suit the capabilities of the software product with minimal customisation.

The main approach that was decided on was one where the business process was also part of the overall system being designed (and not an external requirement to be defined). This meant that in the approach, there would need to be time set aside to design, implement and test the new business processes together with any software being used to support those business processes.

Third, action taking. After deciding to use this agile enterprise system design approach, the team tried using this approach for a period of three months. As any agile approach is an

empirical i.e. “black box” system development method [15], where the analysis, design and development processes in the Sprint are unpredictable (lacking predictable process steps), the team used control mechanisms in the form of agile “artefacts” and “ceremonies” to control risk and manage unpredictability. These ceremonies and how they are conducted are illustrated in table 3 below. The team also used “retrospectives” to continuously review and refine the process over the period of three months.

No.	Artefact / Concept	Purpose
1	Product Backlog	Holds all the business and technical requirements used to develop the system.
2	Epic	Denotes product backlog items of the largest “size”. Usually used to hold a feature set or business workflow – e.g. filing an interaction note after a customer call. May not be extremely clearly defined, but the concept of which is generally understood by the team.
3	Task	A specific activity that comes under an Epic. The description of the task is used to denote business tasks, or technical preparation work that precedes the building of a user story.
4	User Story	A specific activity that involves development or configuration work in the software system. User stories are written in the general format “As a <type of user>, I want to <do something>, so that I can <achieve a purpose>. In system engineering terminology, this is a system requirement.
5	Acceptance Criteria	Every user story should have acceptance criteria framed in the following manner from behaviour-driven development (BDD): a. Given – the beginning state of a scenario b. When – the specific user action c. Then – the outcome of the action This also forms the basis of testing to verify that the user story has been developed accurately.
6	Definition of Done	A more general concept of which acceptance criteria is a specific instance. This is a critical concept to understand whether a

		particular task or user story has been completed according to its original intent (requirements verification). To break down epics into smaller tasks that can be completed in a sprint, the definition of done denotes the extent to which a task should be completed. For example, writing the first draft of a memo for procurement of the software, but not (yet) securing the necessary approvals.
7	Sprint	This represents a fixed-period "time box". In this project, the sprints were one-week long. Each sprint should have the same ceremonies.
8	Sprint Backlog	This is a subset of the product backlog. The sprint backlog determines what is to be done by the team for that particular sprint. It is prioritised in order of importance with the most important item at the top. Sprint backlogs should only have tasks with a clear definition of done, or user stories with clear acceptance criteria.

Tab. 2: Agile Artefacts and Concepts in Project

No.	Ceremony	Purpose
1	Pre-IPM (Iteration Planning Meeting)	<1 hour/sprint> To add new and refine existing backlog items. Items are prioritised so that more important items are at the top of the backlog. High priority items are also broken down into smaller items with more detailed description that allow team members to carry out the fulfilment of the task or build the user story. One of the key aspects of breaking down a backlog item includes specifying the acceptance criteria, or "definition of done".
2	IPM	<1 hour/sprint> To finalise (i.e. get the team's commitment to complete the tasks/stories in) the sprint backlog for the sprint. Important to ensure that the definition of done is clear to whoever is undertaking the task or development of the story.
3	Stand-Up	<Daily> This is where the team meets every day for each member to give an update on: a. What was done in the previous day b. What will be done in the coming day

		c. Any help needed from other team members For this project, the daily stand-up was conducted in written form over Microsoft Teams, a team collaboration platform.
4	Demo	<When-Needed> A demo is done with users or team members to ensure that user stories have been developed accurately. For this project, there was no fixed-time demo session for each sprint, but all user stories had to be checked/tested and accepted by the product manager before release into the production environment. Users were then either informed of the changes or new features, and if the changes were significant, user training would be conducted in order to ensure users are familiar with the changes.
5	Sprint Review	<part of IPM> This is used to review what was done during the sprint and verify what was completed. In this project, because the sprints were short, the sprint review was merged into the IPM, such that the first part of the IPM is used as a sprint review and the second part is then to finalise what should be done for the coming sprint.
6	Retrospective	<1 hour/sprint> This is time set aside for the team to think about and discuss what went well, what did not go so well, and what should be done to improve in the coming sprints.

Tab. 3: Agile Ceremonies in Project

The fourth (evaluating) and fifth (specifying learning) stages of action research will be covered in the third section of this paper on observations, analysis and learnings. It is important to recognise that an agile project is both iterative and incremental, so the action research cycle is also fulfilled within each sprint, and not just at the end of the 3-month period. For example, the sprint retrospective each week gives the team an opportunity to evaluate their actions, derive learnings, and try to implement new ways of doing things to address the problems in subsequent sprints. In this sense, each sprint has the action learning cycle of diagnose-plan-action-evaluate-learning, and the entire 3-month project also part of the action stage of action learning, with diagnose-plan coming before the start of the 3-month project, and evaluate-learning coming afterwards, consolidated in this paper.

C. Research Question and Hypothesis

Besides the business objective of this project, the research question this paper hopes to answer is "is agile enterprise system design a better approach than plan-driven enterprise system design?" The hypothesis is that designing an enterprise system using an agile approach improves overall system quality attributes. In other words, the resulting system that is the output of the project should be of better quality. However, as this is a single case study, the difficulty comes in benchmarking what the output is better than. We have already argued earlier that "organisations can be legitimate objects of scientific inquiry only as single cases", so this organisational system-of-interest (SOI) can only be compared to itself. Ideally, it would be compared to the counterfactual version of itself if the team had used an alternative approach. But this is not practically possible, so we will use a combination of customer evaluation and team self-evaluation. For the customer evaluation, we will ask for an overall evaluation based on the project output versus the customer expectation. For the self-evaluation, there will be two parts. We will give an overall evaluation as well as compare the output to how the team assessed their performance before the start of this project. Both of these evaluations will be conducted using quantitative and qualitative survey methods. We will also observe and evaluate the project proceedings and outcome and provide an assessment from a research perspective.

D. Research Measures

How the output of the project is "better" using an agile enterprise system design approach will be measured in six dimensions. Usability, efficiency, speed, system interoperability, maintainability and cost. Usability refers to the user experience – i.e. how easy it is to use the product. For example, is the software system intuitive and easy to navigate or do users tend to get lost in a deluge of features. Efficiency refers to the business process – does the use of technology make users more productive in performing the required business tasks (through the number of clicks or time spent to complete a workflow). Speed refers to the time taken to complete the project or implement certain features or aspects of the desired business process. System interoperability refers to how well the system works or integrates with other existing systems, especially where the systems are part of a continuous workflow. For example, system interoperability is weaker when data needs to be manually duplicated across systems, or similar tasks need to be repeated across different systems. Maintainability refers to the operating

burden – how easy it is to continue using or supporting the use of the system. For example, if software needs to be continually updated with patches, or if code needs to be refined or change with each new version of the system. Finally, cost. This refers to both cost of build and cost of ownership (overall cost to maintain the system).

E. Project Domain and Team Structure

This particular project domain is in customer relationship management (CRM) for a business-to-business (B2B) organization. The organisational structure covers different units from marketing, sales and service, and within sales, there are also different types of sales teams that have varying practices based on the customer's geography (where they are located) or industry (which industry they belong to). In terms of the business processes covered, the full scope is large and spans different types of engagements with customers. As this is a B2B organization (anonymized), their customers are other organizations, but the business engagements are fronted by employees from the organization, who may attend marketing events with many attendees, or meet directly with sales people in a one-to-one setting.

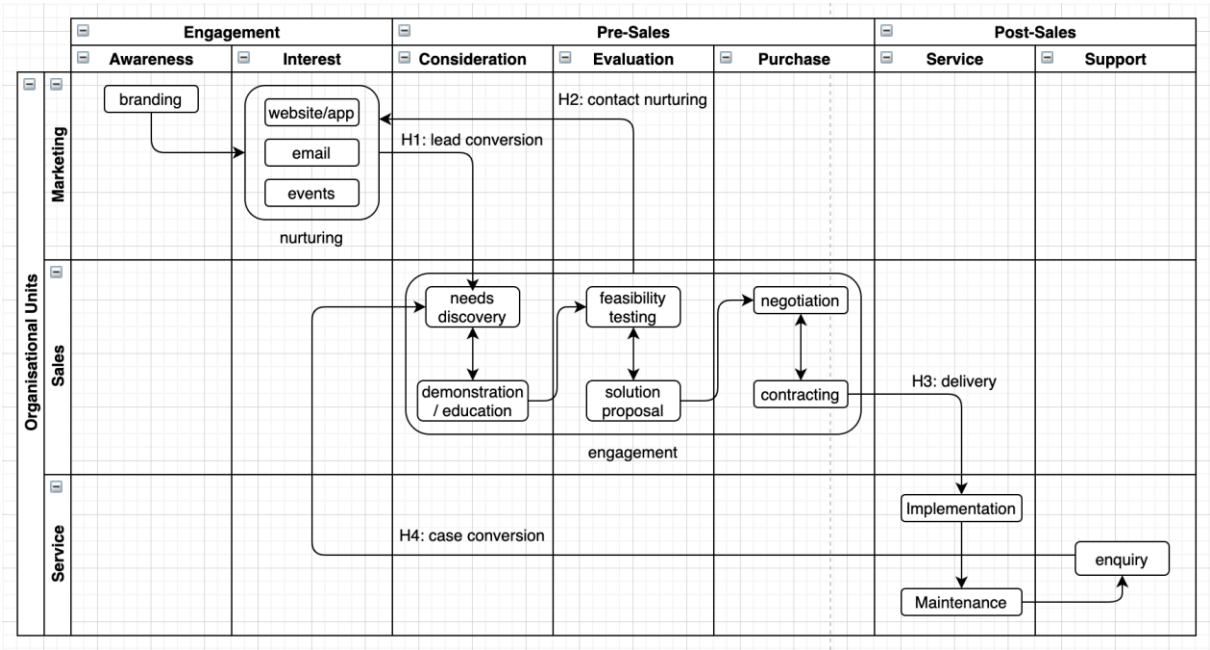

Fig. 5: View of CRM Domain

Figure 5 provides a view of this CRM domain that covers the organizational structure (swim lanes for each organizational unit), the business process (boxes and groups of boxes denoting key business activities) and the flow of information (arrows between boxes). It also highlights

major “handoffs” between different organizational units, H1 to H4, which represent cross-functional collaboration and the need for standard interfacing between components of the enterprise system. Within each department, there are different business processes involving customers and different ways of structuring the information on the customer, so there needs to be agreement between the departments on how to handle the joint business process and information. Using H1 as an example, in lead conversion, marketing is motivated to ask sales to follow up on every interaction, but sales have to prioritize their engagement of leads based on how ready they are to purchase. Both sides have to agree on how “warm” a lead is before marketing should handoff the lead to sales.

The team undertaking this project was small, with only six members. There were two product managers, one responsible for the business aspects, the other from information technology (IT), responsible for the technology implementation. There was a designer and a business analyst, and a technical lead (also doing development) and an infrastructure engineer. At least four members also had subject matter expertise in different parts of the overall CRM process based on their past roles in the organization. Decisions were jointly made by the team through the agile ceremonies outlined earlier, but the business product manager had the overall decision-making authority especially where there were difficult trade-offs to be made. The internal project stakeholders that the team worked with represented the different departments in the organization. There was no wholesale implementation of any particular “flavor” of agile, but the primary influence came from Scrum and the methodology that Pivotal Labs (now known as VMware Tanzu Labs) used, which is a combination of Kanban and XP. Scrum was used as a few members had experience working on Scrum projects previously, while the Pivotal Labs approach was used during team onboarding and training, and also used by other teams within the organization.

F. Agile Enterprise System Design Approach

There are two key aspects of agile enterprise design that this paper intends to observe and analyse. The first is how holistic enterprise design can be done, not just in a predictive, iterative or incremental manner (as described in table 1), but through an agile approach. The second is to use agile enterprise architecture strategies to create a working architectural prototype of the enterprise system.

Holistic enterprise design needs to cover at least organisation, process and information in addition to technology. These are views covered in the enterprise design literature – for example from Giachetti – "The enterprise architecture specifies an enterprise-wide view of the processes, information, and organization of the enterprise and how the three views are integrated. Whenever, a small enterprise project is embarked on, the project deliverables should conform to the enterprise architecture" [7] or in Zachman, which has views for "Data", "Process" and "Network" [30]. As explained earlier, most agile approaches focus more on the software or technology aspects, and treat the organisational structure, business process and information structure and flow as largely external context to the system-of-interest (but perhaps with some degree of influence). As such, in this research, we will pay special attention to how the four aspects are designed in an agile manner. In particular, the way in which the design of the SaaS sub-system is done has a significant impact on the other three aspects. By agreement with the team that during the three-month time-frame, only configuration and no SaaS software customisation will be done, the effect is that business process and information flow aspects of the system may need to be redesigned in order to fit what the SaaS system has been designed to accommodate. This is a key difference in approach compared to most SaaS system projects where business process or information flow requirements are determined first, and software customisations are made in order to support existing or desired business processes. Instead, the project team has to consider the best practice business process supported by the SaaS system early on in the business process and information flow design process. For this project, Salesforce was used as the SaaS system. It was chosen because it is the pioneer in SaaS software and the overall market leader for CRM SaaS systems. It is one of the most configurable systems available, with many features and automation tools provided as configurable instances. So it should be noted that the set of available options even without any software customisation (i.e. coding) is very extensive and not considered significantly limiting.

Figure 6 below is a summary of the first aspect of agile enterprise design. It shows the main difference in approach between the proposed agile enterprise system design approach versus the traditional software or agile software design approach. For enterprise system design, the technology, business process, organizational structure and information structure and flow are all components in the system of interest and can be designed (or redesigned) to meet the overall system requirements. In the traditional software design, or even in the more modern agile software design approach, business processes, organizational structure, and information

structure and flow are considered to a large extent external context, or exogenous to the system of interest, and things that the system needs to interact with.

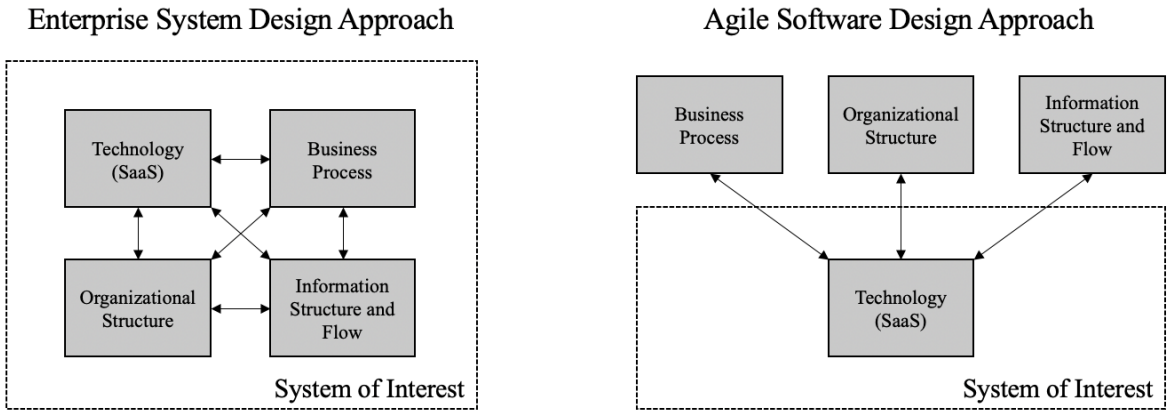

Fig.6: Context Diagram Illustrating Main Difference with Enterprise System Design Approach

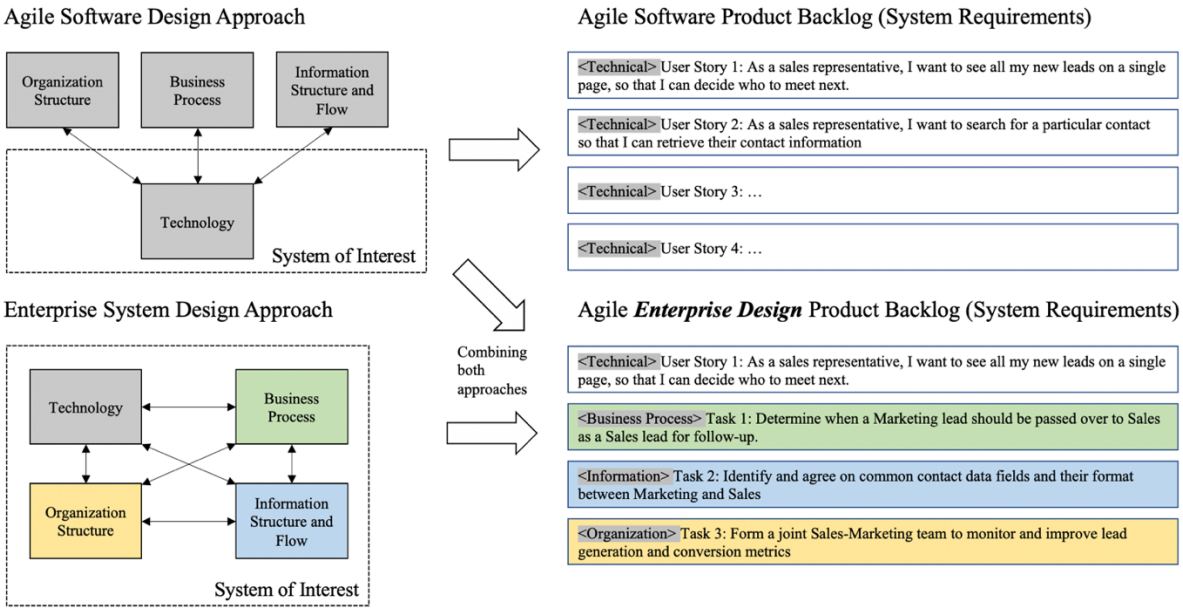

Fig. 7: How the Difference in Approach Affects the Definition and Management of System Requirements

This difference in approach will have an impact on how requirements are defined and managed. In the agile software context, the product backlog is the main artefact that contains the project requirements, with Epics at the highest level, then going down to features and user stories as the smallest grain. Each User story also has a "definition of done" or acceptance criteria to verify the successful development and testing of that story. All of these specifications of requirements pertain to the technical aspect of building the software system. In an agile

enterprise system design approach, we borrow the same artefacts and approach from agile software development, but because organizational structure, information structure and flow and business process are also part of the SOI, we need to have activities (higher-level) and tasks (lower-level) that correspond to the requirements of these aspects of system design. Figure 7 above illustrates how the approach affects the agile backlog in agile software development versus agile enterprise design. An actual example of the agile enterprise design project backlog is also articulated in Table 4 in the Observations, Analysis and Learnings section of this paper.

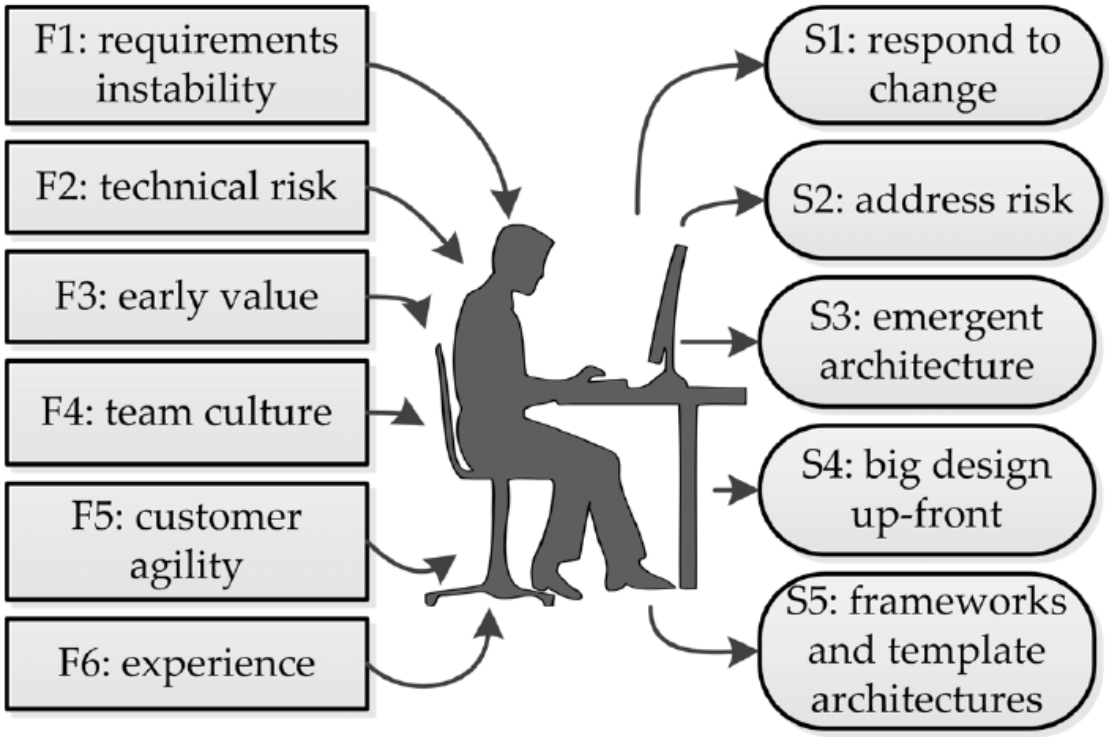

Fig. 8: Forces and Resulting Strategies in Agile Architecture [31]

The second aspect regarding the use of agile EA strategies is based on our discussion that EA needs to be done in an agile manner. This means just-in-time or just-enough EA, and following advocated agile practices such as emergent design or deferring architectural decisions on the basis that the context and resulting needs may change. This does not mean that there is no EA being done, but that the approach to EA is not BDUF. A common understanding of what is to be built is still needed, and working product is still preferred over extensive documentation (for example the many different views of traditional EA). Agile software architecture practices are still in their infancy, let alone Agile EA practices. We borrow the use of "strategies" and

"tactics" from Waterman, Noble and Allan [31], who provide an extensive overview of what agile architecture might look like.

Because there is no prescribed process, they identify driving forces and the resulting strategies and tactics used in response to the context (summarized in figure 8 above). We combine this with the different dimensions of agility from Kotusev [17], who emphasises a sliding scale between full agility and total planning, detailing what is done in an agile versus rigid EA practice (summarised in figure 9 below).

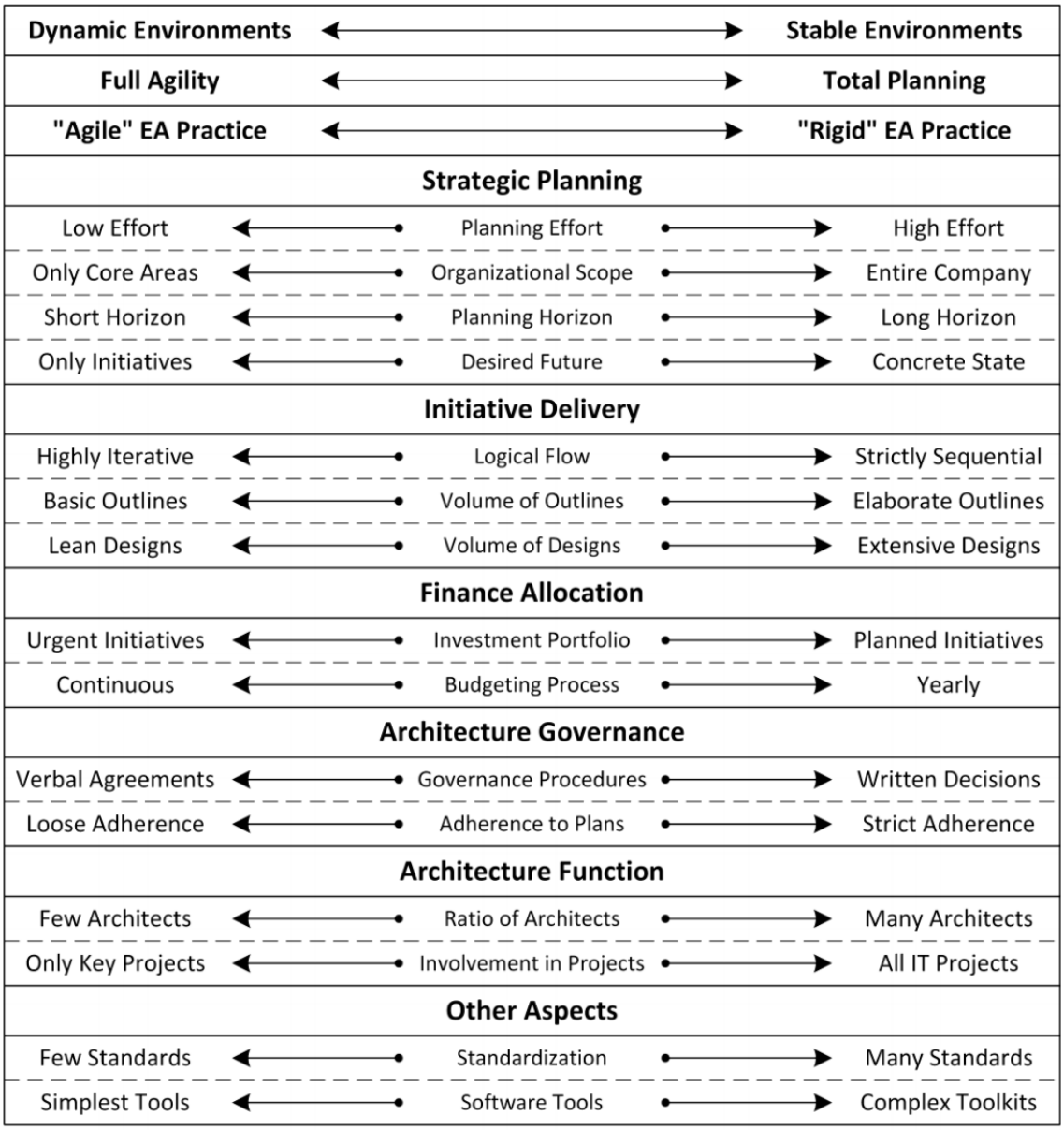

Fig. 9: Different Dimensions of Agility in EA Practices [17]

For this project, the output of agile enterprise design is an architectural prototype of the eventual enterprise system in the form of a working SaaS system. The concept of an architectural prototype is based on the architectural prototype ontology in Christensen and Hansen [32], reproduced in figure 10 below.

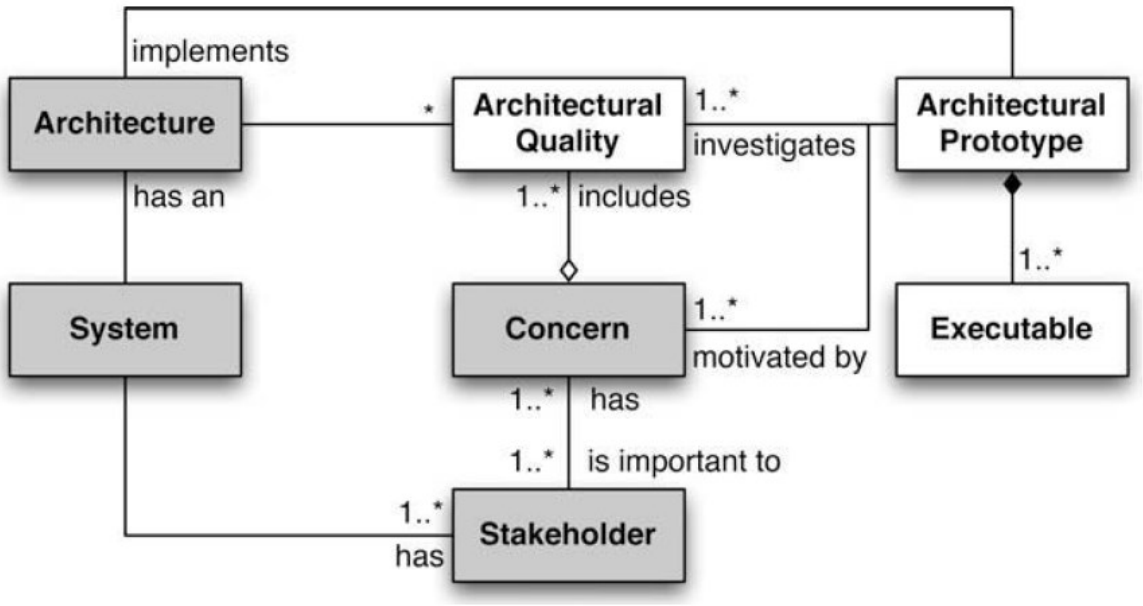

Fig. 10: Architectural Prototype Ontology [32]

The key features of an architectural prototype are that it is executable and it helps to investigate the architectural quality of a system. The prototype will be more extensive than a typical software architectural prototype because the prototype needs to also validate the other three aspects of the enterprise system – process, organisation and information. The SaaS system helps us abstract away the technology layer as the technology infrastructure and detailed design is already contained in a SaaS system – in other words, there already is an existing architecture of the SaaS system that we do not need to detail, but which we can use to help us quickly prototype the technology aspect of the enterprise system without much effort. The output should be working software, used by people in the organisation to perform a business process in order to validate that the enterprise system design works. The reason why it is a "prototype" may be that the number of people using it may be limited, or the business process may have key steps but not all the detailed steps fulfilled, or the data may not be complete, similar to how a software prototype need not have the full set of software features.

Figure 11 below is a summary of the second aspect of agile enterprise design. The top half illustrates the plan-driven enterprise system design approach (taken from Giachetti [7]), where there are distinct stages in the lifecycle, from project initiation to implementation, and specific outputs at the end of each phase in the lifecycle, such as the project charter after the project initiation phase. In contrast, the agile approach does not have fixed lifecycle stages. Instead, there are sprints, and built around those sprints are specific empirical process controls – such as sprint planning, stand-up, sprint review and retrospective (these correspond to the artefacts and ceremonies in section IIb). The key difference is that the activity of architecting in the plan-driven approach takes place within a certain phase in the lifecycle (somewhere between lifecycle stage "Generate and Evaluate Alternatives", and before the full system design is complete in the "Design" stage), and the output being different views of the information, process, organization and technology aspects of the system. In contrast, for agile design, the output of each sprint is an incremental delivery of the system and in the early stages (i.e. during this 3-month project) the increment is an architectural prototype. The square box around both processes represents the activities that are carried out. So for each sprint, there will be activities that span the lifecycle stages of Analysis, Generate and Evaluate Alternatives, Design and Construction in the traditional Enterprise Design process.

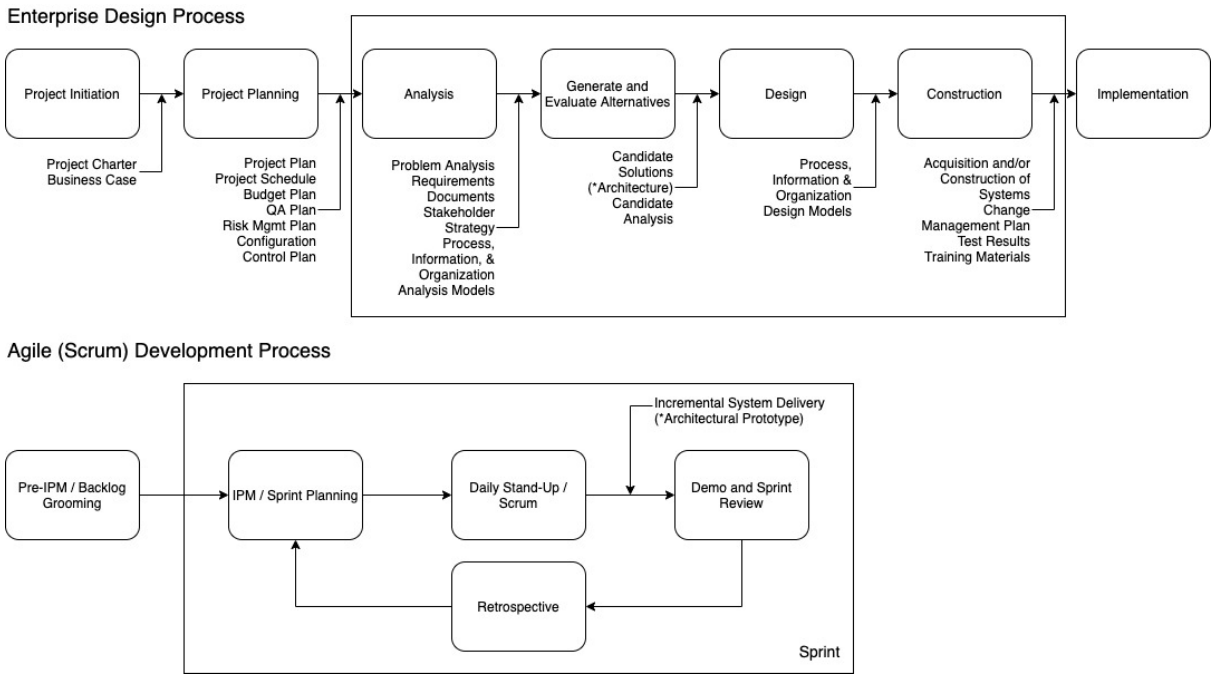

Fig. 11: Plan-Driven Enterprise Design Process Versus Agile Enterprise Design Process

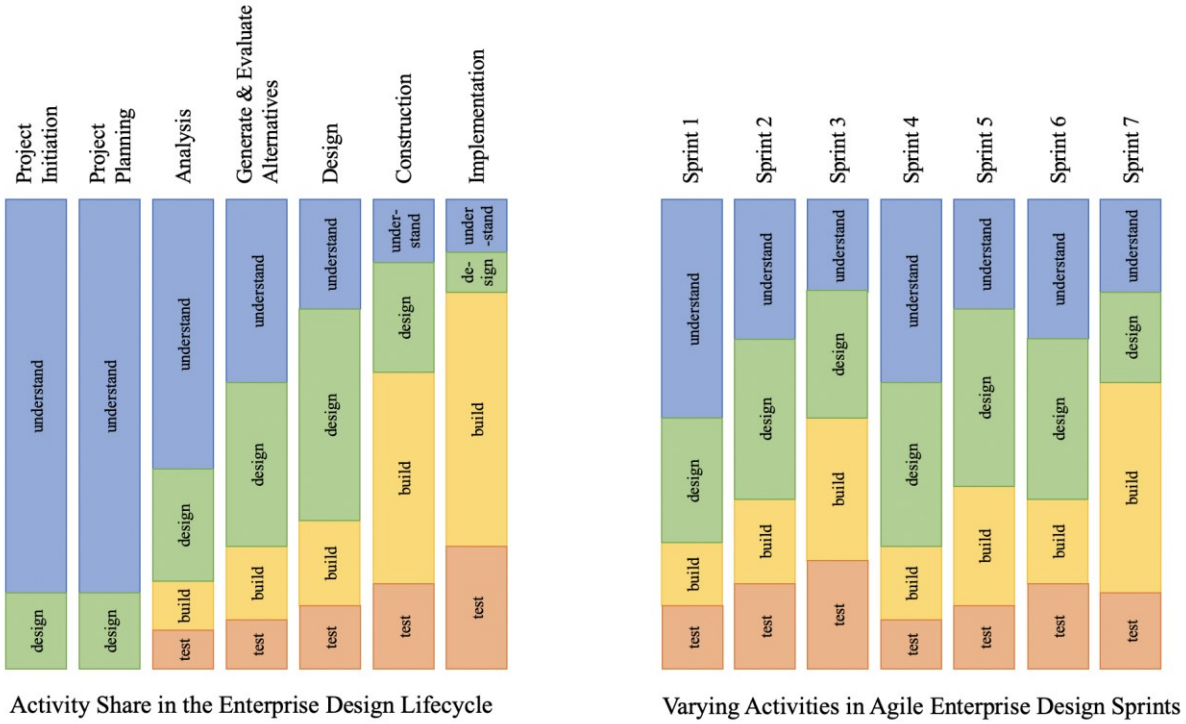

Fig. 12: Pre-Defined Lifecycle Stages and Activity Share in Enterprise Design versus Varying Activities by Sprint in Agile Enterprise Design

Figure 12 illustrates the difference in approach from the perspective of the types of activities performed in each stage of the lifecycle. We group types of activities into 4 basic types – understanding, design, build and test. In the traditional Enterprise Design approach (taken from Giachetti [7]), the lifecycle stages from figure 11 are clearly defined as per a predictive (plan-driven) project management approach. The share of activities is also fairly defined with the majority of understanding activities performed at the onset, and gradually decreasing as the project progresses, while build and test activities gradually increase (similar to traditional systems engineering lifecycles) as the lifecycle progresses. In contrast, the empirical or "black box" agile approach does not have pre-defined lifecycle stages or predetermined activities. Instead of lifecycle stages, there are sprints and the share of these activities in each sprint varies according to the team and the project, as the team determines what activities are needed each sprint to respond to the changing system context. For example, in the first sprint, a small prototype may be built to gather more knowledge about the problem to be solved, and so build and test may already occur then. Because this is a configurable SaaS software platform, there already is a ready system that can be quickly configured (e.g. with clicks, not code) to create that prototype quickly and within a single week versus the time it would take if coding from scratch (i.e. non-SaaS). In addition, as with agile practice, at the end of each sprint, there should

be working software. As such, it is not unusual for every sprint to have some form of build and test activity, even if it is a small feature like adjusting some of the user interface or changing the layout of data fields. Finally, in the agile enterprise design approach, a build/test activity may not always relate to software, but may also be associated with any of the other 3 aspects of the enterprise system. For example, information in the form of picklist data fields may be configured and tested with users to see if they understand the options in the picklist. Such a configuration in Salesforce (the SaaS platform) is fast and easy to configure and change (e.g. under 10 steps).

Figure 13 below is based off figure 11, and is a summary of where the impact of SaaS is felt in the enterprise system design lifecycle. The impact of SaaS in this action research project is therefore (1) the ability to force evaluation of the overall enterprise design ("evaluate whether to adapt business process, information and organization design to SaaS process or vice versa"), (2) the ability to do high fidelity rapid prototyping to understand more about the context and environment (thus improving architecture and design), and (3) the speed of construction (no need for hardware or software installation, and using configuration "clicks", not coding). The added benefit of quickly being able to scale in the implementation phase is not one that is leveraged directly in this action research project, but is a key benefit to the organization once the agile enterprise system design is complete. For example, the organization here has an option to purchase hundreds of additional SaaS licenses to implement the new CRM across the entire sales and marketing organization quickly, and all that is needed is training and change management, rather than more hardware, software acquisition and scaling.

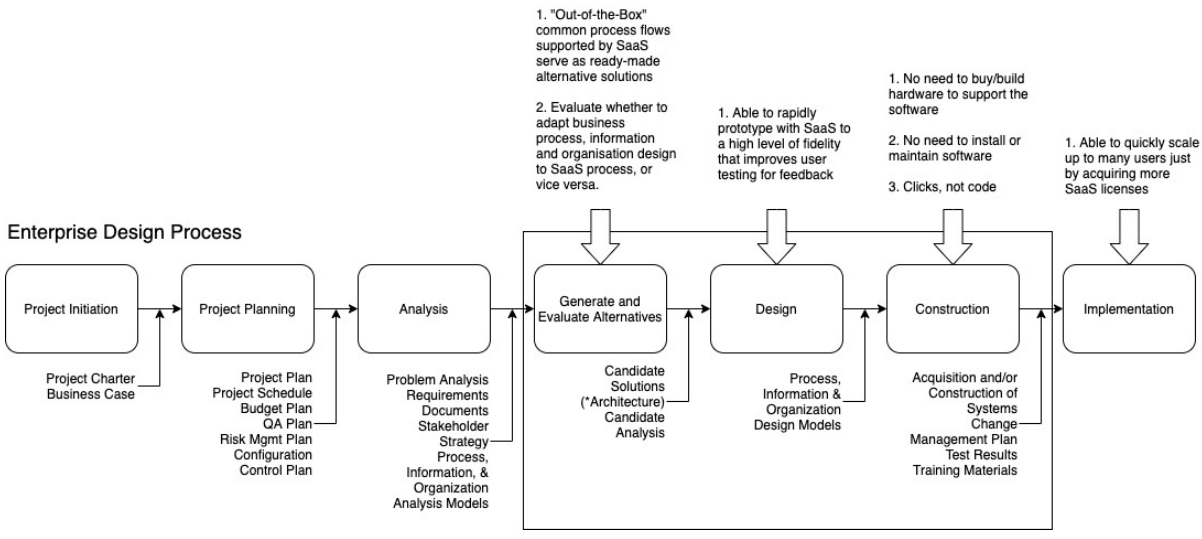

Fig. 13: Where SaaS Impacts the Enterprise Design Process

IV. OBSERVATIONS, ANALYSIS AND LEARNINGS

We re-organise the single case study sources of evidence to support the coming sections. There are three main sources. First, observation, including direct and participant observation, such as participating in the agile ceremonies. Second, documentation, including physical artefacts like the product and sprint backlogs. Third, surveys and one-to-one interviews, which we use to investigate specific viewpoints or areas of interest that might arise from the survey results.

A. Holistic Enterprise System Design

This section focuses on how the project team attempted to do holistic enterprise system design and the outcome of that approach.

The key enabler of this approach was the integrated backlog that included both business tasks and user stories. This was in contrast to how other teams in the organisation used the backlog to only hold user stories for technical (software) development. It was also different to how team members used backlogs in past projects. By integrating both non-technical (i.e. business, information and organisation design) and technical activities and tasks into a combined backlog, the dependencies across these activities were exposed or made "transparent" to the entire team – this is the concept of information radiation in agile practice. An actual example of this integrated backlog can be seen in table 4 below.

No.	Aspect	Task or User Story
1	Business Process	Verify marketing use cases for lead generation and lead nurturing
2	Business Process	Develop storyboards for end-to-end lead generation to lead nurturing business process
3	Technology	Technical "spike" (a form of agile EA exploration) to determine feasibility of integrating Eloqua (marketing system) with Salesforce
4	Technology	Complete preliminary security posture assessment

5	Information	Usability changes to Contact list view and system landing page
6	Information	Get agreement between sales and marketing teams on data fields to be used for lead nurturing
7	Organisation	Propose ownership of sales operations function to be responsible for training, adoption and maintenance of processes

Tab. 4: Example of different enterprise system design aspects undertaken in a single sprint

Table 5 shows the completion status of the backlog over all of the 18 sprints. For each sprint, there is a mix of story type – technical, business, information and organisation, depending on what aspect of the holistic system needed to be designed or reworked. For example, in order to work on a technical story further down the backlog, it may be necessary to first design the business process and how the organisation will be structured around this business process. Process maps might be developed and information flows modelled, and various business teams involved might need to provide input on the process and information flow before the actual prototype is modelled. Alternatively, if the team determined that the process to be designed was completely new and might be difficult for business teams to envisage just based on diagrams, then after a short sprint to design the process independently of the stakeholders, the team might decide to build a small software prototype and ask the various process owners to try out the new process. In this way, the design of the holistic system combines different types of stories (work) over the 18-sprint period.

The chart in Figure 14 shows the burn-up progress of the project over time. From the previous table, it is evident that the technical stories still make up the bulk of the project, versus the non-technical stories. This is to be expected if there are already existing processes and we are not designing something from scratch. Correspondingly, as there already are existing business teams involved in various parts of the sales and marketing process, the number of organisation design-related stories is small. Where they occur, it is largely to work out where new functions of the organisation might be supported (e.g. if a new team needs to be created to support a particular process), and oftentimes, these are major decisions that require significant effort.

Sprint	Stories Completed	Technical Stories Completed	Business Stories Completed	Information Stories Completed	Organisation Stories Completed

1	5	1	3	0	1
2	16	3	12	1	0
3	9	5	4	0	0
4	7	2	4	1	0
5	9	1	3	5	0
6	14	0	9	4	1
7	7	2	4	1	0
8	4	2	2	0	0
9	6	2	3	1	0
10	3	0	1	2	0
11	6	1	5	0	0
12	3	1	1	1	0
13	3	0	3	0	0
14	14	2	8	4	0
15	6	1	4	1	0
16	11	4	5	2	0
17	7	1	4	1	1
18	11	0	8	3	0

Tab. 5: State of backlog – type of stories completed in each sprint

Since this is a customer relationship management system that involves a lot of customer data, we would also thus expect the information design component to be sizeable. What is important to note is that we are able to account for the non-technical design activities in the backlog and in the burn-up chart. This is what is fundamentally different about this holistic agile enterprise design approach. Most agile software projects take the non-technical design activities as exogenous factors, only estimating and accounting for the technical work. As such in typical backlogs, the breakdown type is by user stories, bugs and other related technical issues.

What is not yet seen in the backlog completion table and burn-up chart is the number of stories that were planned but not completed and carried over to the next sprint – i.e. “roll-over”. One of the key challenges faced by this holistic enterprise system design approach is the ability to estimate effort effectively across different types of user stories. For a start, the non-technical work is not typically estimated in this way. In fact, we use the term “stories” loosely in a non-

technical sense, as these may actually be more like business tasks. Breaking things down into fine-grained tasks and accounting for them in a consolidated backlog is not something typically done. Secondly, non-technical stories or tasks may have high dependency on other teams (not fully within the project team's ability to complete) and thus difficult to break down into something that can be completed in a single sprint. Thirdly, even if work breakdown was possible, equitable effort estimation across the different types of stories was very difficult. As this way of working together was very new to the team, at the start of the project, they agreed to focus on work breakdown and tracking first, and defer estimation until a they were more familiar. There were subsequently a few attempts to try and estimate the work for each sprint, but discussions soon broke down because the types of tasks or stories being compared were different in nature.

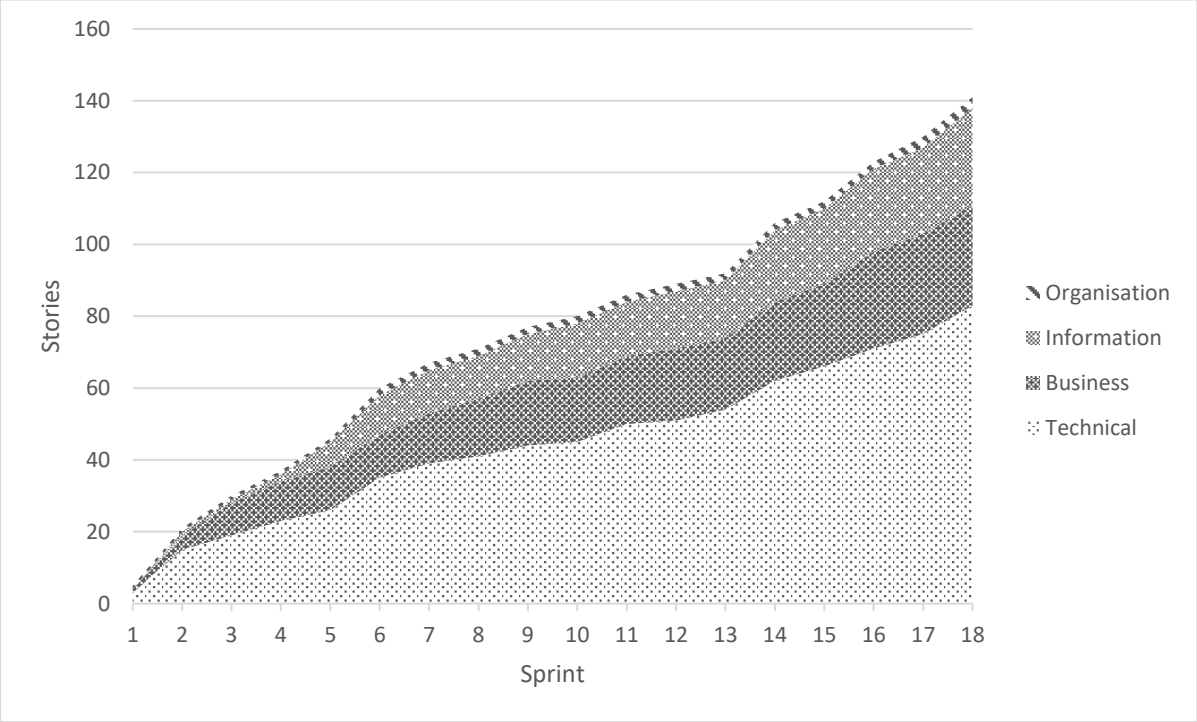

Fig. 14: Burn-up chart – cumulative stories completed with breakdown by type

While this paper still believes it is prima facie possible to estimate effort across different types of tasks (for example, man-day effort from the team required to complete different tasks, even if the "man-day" requires a specific team member or resource to complete), it was observed in practice that formal estimation was difficult. This manifested in the propensity for stories to roll-over, summarised in Table 6.

Sprint	Total Roll-Over Propensity	Business Roll-Over Propensity	Technical Roll-Over Propensity	Information Roll-Over Propensity	Organisation Roll-Over Propensity
1	28.6%	0.0%	40.0%	NA	0.0%
2	5.9%	25.0%	0.0%	0.0%	NA
3	35.7%	16.7%	50.0%	NA	NA
4	53.3%	33.3%	42.9%	75.0%	100.0%
5	55.0%	50.0%	70.0%	28.6%	100.0%
6	22.2%	100.0%	18.2%	0.0%	0.0%
7	41.7%	66.7%	20.0%	0.0%	NA
8	60.0%	50.0%	60.0%	100.0%	NA
9	45.5%	0.0%	40.0%	75.0%	NA
10	25.0%	NA	50.0%	0.0%	NA
11	25.0%	50.0%	0.0%	100.0%	NA
12	25.0%	50.0%	0.0%	0.0%	NA
13	80.0%	100.0%	70.0%	100.0%	NA
14	36.4%	66.7%	27.3%	20.0%	NA
15	62.5%	80.0%	55.6%	50.0%	NA
16	35.3%	20.0%	44.4%	33.3%	NA
17	53.3%	0.0%	55.6%	75.0%	0.0%
18	0.0%	NA	0.0%	0.0%	NA
Total	40.3%	46.2%	37.1%	42.6%	40.0%

Tab. 6: The percentage of planned stories rolled over for each sprint (NA denotes no stories planned)

Based on the interviews conducted, the combined backlog of business and technical tasks/activities was useful in tracking and reporting on progress as a team. Previously, the project was conducted in a waterfall (predictive) manner with parallel workstreams for each function like design or system integration. Compared to this, the coordination was much better and the ability to change dependent aspects improved. In addition, the team was able to try out different ideas on the overall enterprise system design, and execute them in small parts to test them with business process users

However, there were some downsides to the integrated enterprise design approach. For all the team members, it took a lot of effort to agree on how tasks should be defined in the backlog. For example, to what degree, level or size they should be broken down into, especially when they were not user stories. During the team discussions, some team members also explained that they were unused to breaking down their tasks into such detail, and one of the team members even went to far as to say it felt like "micro-management" of tasks. This feeling and difficulty did not go away even towards the end of the three-month period, after more than 12 sprints were conducted. From observation, although the expressed intent of having these tasks and activities detailed in the backlog was so was to understand dependencies and give other team members visibility on what tasks they might have to allocate time and effort to subsequently, some team members did not value the transparency as it meant that they had to commit to meet certain timelines while not being completely in control of the task (e.g. because certain steps had to be done by other people outside of the team). On the other hand, some team members were noticeably more comfortable with this way of detailing their tasks, and accounting for them at the end of the sprint, even if it meant they had to explain why the task was not complete and needed to be "rolled-over" to the next sprint. This seems to depend on personalities and team dynamics. Eventually the team had to work with tasks that were defined to varying degrees of specificity – some with clear definitions of "done" where the completion milestone was objectively evident, and others that were more like placeholders for ongoing tasks, with the definition of done only made clear when the task was near completion.

B. Agile EA Strategies

The other area of investigation was in the use of agile EA strategies in the project. In this regard, there was positive feedback from the interviews in three main areas – dealing with requirements instability, having an emergent architecture and managing scope.

Against the force of requirements instability (figure 8), the team found it was able to use prototyping to discover and validate business needs and requirements. For example, if Salesforce has a certain out-of-the-box way of handling a certain process, the team could use that as a prototype to test with actual users of the business process and get their feedback or response before determining the actual business process design. As a result of this, certain business process features were adopted that would otherwise not have arisen if business processes were first determined in a predictive manner. The prototyping effort also helped the

team handle requirements instability by using out-of-the-box processes in Salesforce as the baseline for undefined or poorly defined business process requirements, and adjusting the process from the baseline in the form of configuration. If the revised process was still not suitable, an interim process was used that did not require customisation, with the intent of undertaking the actual software customisations subsequently. By using the prototype over a period of time, the team also found that some of the requirements were not necessary as users got used to doing things in a different manner. All this was in sharp contrast with how the project was conducted before, where there was a lot of anticipation and guessing in coming up with what should be a complete set of business requirements for a subsequent development phase in the waterfall approach.

There was also an emergent architecture that the team started to have a common understanding of as the prototype took shape. In the previous plan-driven approach, knowledge of the overall architecture was held by certain team members who were planning it, and because the scope was extensive, it was difficult to plan upfront as different team members had different abstract concepts in their mind. By working in increments and having a physical, working prototype to rely on, the team was able to learn collectively what the gaps were and what needed to be done next.

Finally, in the area of scope management to deliver early value, we saw previously that the domain of CRM was very extensive, so working in an agile manner forced a strict prioritisation of effort and helped the team to be clear about the deliverables for each sprint. For example, more important epics would form the basis for immediate sprints, and within each sprint, the tasks were also ordered by priority. As there were many systems to integrate with, this agile EA approach also forced the team to decide on which integrations were critical and to set aside time to investigate how they could be done. Before this, because there were so many different departments with different stakeholders, the problem statement was pulled in different directions and the scope kept increasing (i.e. “scope creep”).

While the above were clearly positive effects of the agile EA approach, there was one downside in that this agile EA approach was difficult to align with the non-agile architectural planning of the rest of the organisation. For example, extensive integration plans across all the different systems were being developed in parallel and would require allocating developer resources from the different project teams to develop the application programming interfaces (APIs) for

integration. Some of the technical "spikes" were also not possible to complete without the help of specific engineering resources (for example, testing the integration with systems owned by other groups in the organization).

C. Validation of Research Hypothesis

We use the results of surveys with team members and stakeholders to try and validate the research hypothesis that that designing an enterprise system using an agile approach improves overall system quality attributes. Validation is difficult because architecture is unseen and its quality attributes sometimes only manifest themselves over time, and in relation to specific contexts. We therefore use the team members opinion/views on the approach as a form of internal validation, and the project stakeholders (internal customers) opinion/view as a form of external validation. For the team members, they are asked to assess the project approach on a scale of one to five, as well as relative to the previous predictive (plan-driven) approach. For the stakeholders, they are asked to assess the approach in relation to their expectations of how the project should have gone. The details or rationale behind the scores were elicited through group discussion of survey results or in the subsequent interviews.

	Overall Product Quality	Usability	Business Efficiency	Works with Other Systems	Speed
Max Score	4	5	3	3	4
Min Score	2	1	2	1	1
Spread	2	4	1	2	3
Average Score	3.3	3.5	2.6	2.3	2.4

Tab. 7: Overall Rating of Approach (Team Self-Assessment)

The summarised scores for the team member assessment are in table 7 above. This is the overall project rating given on a Likert scale (1 to 5), with 1 as the worst, 5 as the best, and 3 as average. The agile enterprise design approach yields an above-average product quality and usability (average score of 3.3 and 3.5), but it was acknowledged that the system does not work very well with other systems due to a lack of integration, and business efficiency was slightly

compromised as a result (average scores of 2.3 and 2.6 respectively). This is not so surprising as a result – a common issue associated with agile development is that it generally works well with small projects but has difficulty scaling to take into account the broader enterprise architecture. The ability to move fast (average score of 2.4) was also affected by the pace at which the organisation-wide integration planning was proceeding (in a non-agile manner).

Despite the strictness of the team overall self-assessment, the relative assessment between the new approach and the previous plan-driven approach was more positive, and reflected an improvement in approach. This can be seen in table 6 below. This relative rating also used a Likert scale of 1 to 5 with 1 being much worse than the previous approach, 5 being much better than, and 3 being no better or worse than the previous approach. Compared to the overall assessment, the relative assessment had a spread that was not as wide, and notably, the minimum scores were at least a 3 for the measures of overall product quality, usability, speed and business efficiency – this meant that all team members assessed that the new approach was at least as good, if not better than the previous approach in these 4 measures. In addition to this, on average, for all 5 measures, the new approach worked at least as well, if not better. This is a clear validation by the team of the agile enterprise system design approach, after trying it out for three months. Further validation is seen in that the team decided to continue with the approach after the action research period, thus suggesting that it worked well for them. In terms of the minimum score, two respondents indicated a lower score for "Working with other Systems". They felt the deliberate up-front planning approach was better in this regard because of the overall planning done and the compatibility with the non-agile planning approach used by the rest of the organization. This was elaborated on through the interviews (summary in table 10).

	Overall Product Quality	Usability	Business Efficiency	Works with Other Systems	Speed
Max Score	5	5	4	4	4
Min Score	3	3	3	1	3
Spread	2	2	1	3	1
Average Score	3.8	3.8	3.8	3.0	3.4

Tab. 8: Relative Rating of Approach (Team Self-Assessment)

On the remaining two research measures of cost and maintainability, these can be compared between the original plan-driven approach versus the agile approach. For maintainability, it is argued that because the agile approach used only configurations in the architectural prototype, the resulting system is more maintainable as compared to the output from the plan-driven approach, which intended to specify all the business requirements upfront and hire a technology service vendor to develop and implement the software solution. Because of the sequential, even if iterative, nature of the plan-driven approach, many requirements may have been specified that might not actually be needed, as we found out during the course of the agile project. Because of the lack of holistic enterprise system design, there would also be less of an opportunity to design business processes with the SaaS-supported best practices as a baseline. Even if eventually the service vendor managed to work together with the team to redesign business processes, the agile approach would be at least as good, if not better than the plan-driven approach, in terms of using configurations rather than customisation. To the extent that the system is customised, maintainability is affected as Salesforce releases major system updates three times a year, and each time a release occurs, existing customisations face the risk of not being compatible and needing to be reworked.

For the cost measure, the agile approach can again be argued to be better. This is because prior to the agile approach being used, the team had tried to solicit vendor quotations for the initial set of business requirements. Due to the lack of specificity in requirements – e.g. missing related requirements, a lack of details for many requirements, and the fuzziness of scope boundaries, the initial vendor quotations were extremely high. This was to buffer for scope creep and rework, time and effort needed to work with business stakeholders on requirement details, and to specify or fill in any missing requirements. The total manpower cost of the 6-member team for the three-month period did not amount to anywhere near that quotation, and the scope of work completed was similar, if not superior, in terms of delivering a higher amount of value in the form of working product.

We now turn to external validation by the project stakeholders who are also the internal customers of this project. We use a combination of surveys and interviews. Similar to the team assessment, we ask for the customer assessment along the five measures, also on a Likert scale of 1 to 5. The difference between the team assessment and the customer assessment is that the

customers were only involved specifically for the agile enterprise design phase of the project, so they were not able to directly compare their experience with a previous, non-agile approach. This means that we were only able to collect their overall rating of the approach, and not a relative rating (as with the team assessment).

	Overall Product Quality	Usability	Business Efficiency	Works with Other Systems	Speed
Max Score	5	4	4	4	5
Min Score	4	3	3	3	3
Spread	1	1	1	1	2
Average Score	4.3	3.7	3.7	3.3	3.7

Tab. 9: Overall Rating of Approach (Stakeholder Assessment)

As with the team assessment, the challenge with an overall rating is that it is influenced to different degrees by both individual expectations of the project as well as the extent of prior or concurrent experience in other projects (regardless of whether they are similar in nature or not). This key difference means that we may only be able to use the stakeholder assessment more as supporting evidence or indirect validation rather than direct validation of the approach. For the results, all of the ratings along each measure were at least average, if not higher, and the spread was small, perhaps with the exception of speed. However, on that measure, the interviews suggested that the issues with speed were more about the time it took for "alignment" with other stakeholders – i.e. agreeing to participate in the project and changing their existing business process. The stakeholder that gave the highest rating on speed felt that progress was very fast in spite of the difficulty in stakeholder alignment, suggesting that the agile approach did improve the pace of progress despite difficult external context. The lowest average score was in interoperability, or how well the new/revised business process and supporting system fits with other business processes and systems. For some of the lower scores, the stakeholders explained that this was an interim rating because firstly, the users on the agile project still needed to interface with processes and systems used by other teams who were not yet on the project, and secondly, the time-frame for testing and assessment is relatively short and therefore difficult to state definitively. This feedback is consistent with the team self-

assessment discussed earlier, where working with other systems was also the lowest scoring area in the overall assessment.

The feedback from interviews has been integrated into the sections above, but it is worth spending some time to look at the themes that emerged and the qualitative input from both project members and customers. Interviews were conducted individually, and with standard questions on challenges with both approaches, and specific practices that were helpful or unhelpful in addressing the challenges. Additional questions were calibrated to get better understanding of specific topics based on the participant's role, function or feedback. Table 10 is a summary of the responses according to key themes. The agile enterprise design approach clearly fared better in the areas of (1) dealing with requirements instability, (2) creating emergent architecture, (3) managing scope. This is consistent with the benefits of agile enterprise architecture and is the result of employing agile enterprise architecture strategies and tactics outlined in Figure 8 – S1: Respond to Change and S3: Emergent Architecture. Themes 4 and 5 were less clear in terms of absolute benefit. The plan-driven approach was found to be lacking, but although there were benefits to the agile approach, these were qualified with some downsides as well, such as difficulty aligning with the rest of the (non-agile) organisation, or agreeing on the right level to define tasks in the backlog across technical, business process, information and organisational stories. In fact, from observation, there were numerous discussions and debates over this, which consumed a large amount of the team's time. Themes 6 to 8 were areas where the plan-driven approach was still found to be better. The main challenges in team coordinating were adapting to an approach that was new to the team. It is difficult to say just from this experiment alone, if the newness of the approach was a factor, and something that might diminish over time as the team got more used to it. However, it is noted that the combining of different types of components in enterprise system design is a cause of the process complexity, as one of the participants who had previously run agile software development projects commented that in such an approach, they only had to define and manage technical user stories (which by themselves were already challenging to manage). This undermined the self-organising team dynamic required for agile projects, as the product manager often had to take the lead to bring project members along with the approach. Finally, in theme number 8, the team eventually acknowledged that for the purpose of stakeholder management and aligning the team with the end-state vision, it was still important to provide longer-term timeline planning, versus looking ahead a few sprints at a time. This was something that was adopted and integrated into the agile approach, as per the plan-do-check-

adapt cycle. As such, it could be considered part of the agile approach after all, since only process controls are defined in the empirical approach, and it is up to the team to define how they want to actually run the agile project.

No.	Theme	Previous (Plan-Driven) Approach	Experimental (Agile) Approach
1	Requirements Instability	Unclear/ambiguous business problem needed "educated guessing" and "figuring out"	Able to use drafts/prototypes to discover and validate business needs and requirements
2	Emergent Architecture	Difficult to plan upfront as different people had different abstract concepts in mind	Worked in increments for team to discover what needs to be done
3	Scope Management	Stakeholders "pulled problem statement in different directions". "Scope Creep" – i.e. scope kept increasing	Had strict prioritization, and was clear about deliverables for each sprint
4	Enterprise Architecture	Planning was done, but not executed as it was too complex/extensive	Able to execute in small portions and test/try out different ideas. But difficult to align this approach with non-agile architectural planning of the rest of the organization
5	Team Coordination	Team was split into parallel tracks, so it was hard to optimize as a group	Combined business and technical backlog was useful to track and report on progress, but took effort to align on how tasks should be defined in the backlog (e.g. at the right level/size)
6	Team Coordination	Less confusion around roles and responsibilities with individuals leading separate workstreams	"Self-organizing" team dynamics were more ambiguous. Burden seemed to fall on the PM to drive the project and coordinate

7	Team Coordination	Simple, straightforward way of working	Stumbled over many "technical" terms and details. Sometimes lost sight of the intent.
8	Product Vision	Longer-term timeline helped manage stakeholder expectations and focus team on longer-term objectives	Near-term focus and some members felt micromanaged. **eventually adopted timeline approach as well*

Tab. 10: Main Themes and Supporting Feedback from Interviews

A summary of the performance of the agile enterprise system design approach as compared to the previous plan-driven approach along the six research measures is in table 11 below.

Measure	Outcome	Form of Validation
Usability	Agile	Direct team assessment, indirect stakeholder assessment
Efficiency	Agile	Direct team assessment, indirect stakeholder assessment
Speed	Agile	Direct team assessment, indirect stakeholder assessment
Interoperability	Unable to determine	Direct team assessment, indirect stakeholder assessment
Maintainability	Agile	Empirical reasoning
Cost	Agile	Empirical reasoning

Tab. 11: Comparison between Plan-Driven and Agile Enterprise System Design Approaches.

The agile enterprise system design approach is determined to be better than the plan-driven approach in five of the six predetermined measures. The form of validation for the first four measures was primarily through the relative assessment surveys by the project team members and their interview feedback ("direct team assessment"). This is "direct" because the team was able to compare directly the plan-driven approach used before the start of the action research phase, to the agile approach used throughout the three-month experimental period. In addition to this validation is the feedback from customer stakeholders, which is "indirect" because it does not directly compare the outcome from the agile experiment to the plan-driven approach. This was not possible because the customers did not see what was going on behind the scenes, and were therefore not able to compare the approaches directly. They could only see the outcome at the end of the three-month action research phase and rate that outcome. It should

be noted that the customer survey scores were higher than the team's absolute survey scores, so it is interesting that the team seemed to be harsher on their own performance compared to what the customers saw as an above average outcome (scores above 3 on a Likert scale). The last two measures of maintainability and cost are validated based on empirical reasoning already argued earlier in this section.

D. Shortcomings and Future Improvements

The main obstacle to the applicability of the results of this research is that it is based on a single case study. We have previously argued that this in itself is not a shortcoming, because each organization is unique and "organisations can be legitimate objects of scientific inquiry only as single cases". However, we recognise that for the sake of generalisation of results, and knowing when it might be good to apply this approach to future cases, this research would benefit from additional case studies with similar projects or similar organisations, so that there is some degree of comparability. To replicate this research across more cases studies is a difficult task because the ability to contract on such action research studies is challenging, made even more so by the need to contract at the right time (just as a change in approach is being considered) and for the right kind of project (one with an extensive enterprise design mandate and not just a software development goal). There are also challenges with comparability, such as acknowledged in Galster et al. [33]. Nonetheless, this is a meaningful approach to further extend the scope and credibility of the results. Another way to improve this research would be to try and prove the hypothesis using a broad approach across multiple case studies. This would mean less detail for each project, but acquiring a large enough number of similar projects to perform some degree of statistical analysis to prove (or disprove) the hypothesis. The difficulty with this is being able to collect the same data across all these projects, who may not be proceeding at exactly the same time, and may have varying practices (we will need to determine if their practices qualify as agile enterprise design or not). There has yet to be a credible research paper using this approach even for agile software development, which is more established than agile enterprise system design. Most of the papers cited either use a single case study approach [2] [23], or conduct surveys and interviews of teams that claim to be practicing agile software development [27].

V. CONCLUSION

While enterprise system design is holistic in its identification of components that contribute to the overall system, the approach used is largely predictive and iterative at best, not well equipped to address the ill-defined and complex nature of enterprise design problems. On the other hand, Agile software development is designed to handle change and complexity, but focuses mainly on technical system design and considers the other holistic enterprise design aspects – organization, information and business process – as largely exogenous, i.e. something the project team has to respond to as part of external change. This is not a realistic starting point, and may lead to sub-optimal system design outcomes as it is often not possible to define at the start of the project, what the system boundaries are, and decide what is exogenous or not. This is especially the case if the nature of the problem is ill-defined and complex, which are the very sorts of problems that agile was designed to tackle. If Agile is about being able to design a solution in response to change, and if the business, information or organization aspects are also changing, then it is difficult to respond to these changes if the ability to design such components of the system are not within the project team.

This paper argues for combining both agile and enterprise system design into an agile enterprise design approach. We have illustrated through an action research experiment what this approach might look like in the context of a real-life configurable SaaS software system design project. Through this project, we have demonstrated that it is possible to holistically coordinate business, information, technical and organization design in agile manner in response to change. There was actual product delivered within the observed time frame, and the benefits of this approach was assessed both internally (team members) and externally (stakeholders) to be more effective in the areas of usability, efficiency, speed, maintainability and cost. Based on this research outcome, we conclude that the answer to the research question "is agile enterprise system design a better approach than plan-driven enterprise system design?" is "yes". Designing an enterprise system using an agile approach improves overall system quality attributes. In other words, the resulting system that is the output of the project is of better quality.

However, because this is a single case study and action research approach, there are limits to the generalizability of the findings – i.e. we cannot say this is true in all cases. The answer of "yes" is limited to the context of the case study – e.g. a small team, a medium-sized organization, a complex and ill-defined problem with changing requirements, and a system that needs to be designed across departments and as a subset of a larger enterprise system. Arguably,

this kind of context is not atypical or considered an outlier, so one could assert that there are many similar scenarios where this is generalizable and where this research might be applicable or hopefully instructive (for example, what enterprise system design project is not complex, or does not come with ill-defined requirements?) Nonetheless, we have been critical about the shortcomings, and have proposed possible future areas of research.

VI. REFERENCES

[1] S. Bondar, J. Hsu, A. Pfouga and J. Stjepandic, "Agile digital transformation of System-of-Systems architecture models using Zachman framework," *Journal of Industrial Information Integration,* vol. 7, pp. 33-43, 2017.

[2] B. B. C. L. M. H. Maria Paasivaara, "Large-Scale Agile Transformation at Ericsson: A Case Study," *Empirical Software Engineering,* vol. 23, no. 5, pp. 2550-2596, 2018.

[3] J. Lapalme, A. Gerber, A. Van der Merwe, J. Zachman, M. De Vries and K. Hinkelmann, "Exploring the future of enterprise architecture: A Zachman perspective," *Computers in Industry,* vol. 79, pp. 103-113, 2016.

[4] TOGAF® Standard, Version 9.2, [Online]. Available: https://pubs.opengroup.org/architecture/togaf9-doc/arch/index.html.

[5] D. Nightingale and D. Rhodes, Architecting the Future Enterprise, Cambridge, Massachusetts, USA: Massachusetts Institute of Technology, 2015.

[6] W. B. Rouse, Enterprise Transformation - Understanding and Enabling Fundamental Change, Hoboken, New Jersey, USA: John Wiley & Sons, 2006.

[7] R. Giachetti, "Enterprise Architecture," in *Design of Enterprise Systems : Theory, Architecture, and Methods*, 2010.

[8] J. Martin, "Using Architecture Modeling to Assess the Societal Benefits of the Global Earth Observation System-of-Systems," *IEEE Systems Journal,* vol. 2, no. 3, pp. 304-311, 2008.

[9] M. Fowler, "Who Needs an Architect?," *IEEE Software,* 2003.

[10] I. 42010:2011, "ISO/IEC/IEEE 42010:2011 Systems and Software Engineering — Architecture Description," ISO Online Browsing Platform, [Online]. Available:

https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:42010:ed-1:v1:en. [Accessed 30 March 2021].

[11] E. Rechtin, Systems Architecting, Creating and Building Complex Systems, 1991.

[12] Y. Gong and M. Janssen, The value of and myths about enterprise architecture, vol. 46, International Journal of Information Management, 2019, pp. 1-9.

[13] The Open Group, "Applying Iteration to the ADM," [Online]. Available: https://pubs.opengroup.org/architecture/togaf9-doc/arch/index.html.

[14] Silicon Valley Product Group, "Agile Development Processes," 2005. [Online]. Available: https://svpg.com/agile-development-processes/. [Accessed 19 March 2021].

[15] K. Schwaber, "SCRUM Development Process," in *Business Object Design and Implementation*, London, Springer, 1997, pp. 117-134.

[16] R. T. B. Boehm, "Management challenges to implementing agile processes in traditional development organizations," *IEEE Software,* vol. 22, no. 5, pp. 30-39, 2005.

[17] Kotusev, "What is Agile Enterprise Architecture?," British Computer Society, 2020. [Online]. Available: https://www.bcs.org/content-hub/what-is-agile-enterprise-architecture/. [Accessed 19 March 2021].

[18] C. Alexander, Notes on the Synthesis of Form, Cambridge, Massachusetts and London, England: Harvard University Press, 1964.

[19] G. Schafer, M. Schulze, Y. Yusuf and A. Musa, "Benefits of SaaS-Based Enterprise Systems for SMEs - A Literature Review," in *Decision Support Systems II – Recent Developments Applied to DSS Network Environments*, Liverpool, UK, 2012.

[20] M. Conway, "How do Committees Invent?," *Datamation,* 1968.

[21] P. Kruchten, "Software Architecture and Agile Software Development—A Clash of Two Cultures?," in *ACM/IEEE 32nd International Conference on Software Engineering*, 2010.

[22] B. Boehm, R. Valerdi and E. Honour, "The ROI of Systems Engineering: Some Quantitative Results for Software-Intensive Systems," *Systems Engineering,* vol. 11, no. 3, pp. 221-234, 2008.

[23] G. K. H. T. D. G. A. J. O. N. Torgeir Dingsøyr, "Developing Software with Scrum in a Small Cross-Organizational Project," in *Software Process Improvement, 13th European Conference, EuroSPI 2006*, Joensuu, Finland, 2006.

[24] S. Ambler, "Agile Software Development at Scale," in *IFIP Central and East European Conference on Software Engineering Techniques - Balancing Agility and Formalism in Software Engineering*, Poznan, Poland, 2008.

[25] G. van Waardenburg and H. van Vliet, "When agile meets the enterprise," *Information and Software Technology,* vol. 55, no. 12, pp. 2154-2171, December 2013.

[26] J. M. Bass, "Scrum Master Activities: Process Tailoring in Large Enterprise Projects," in *IEEE 9th International Conference on Global Software Engineering*, 2014.

[27] T. Imani and M. Nakano, "A Model for Effective Area of Hybrid Approach Combining Agile and Plan-Driven Methods in IT Project".

[28] R. Yin, Case study research and applications: design and methods, sixth edition, SAGE, 2018.

[29] R. E. Gerald Susman, "An Assessment of the Scientific Merits of Action Research," *Administrative Science Quarterly,* vol. 23, no. 4, pp. 582-603, 1978.

[30] J. A. Z. J. F. Sowa, "Extending and formalizing the framework for information systems architecture," *IBM Systems Journal,* vol. 31, no. 3, pp. 590-616, 1992.

[31] M. Waterman, J. Noble and G. Allan, "How Much Up-Front? A Grounded Theory of Agile Architecture," in *IEEE/ACM 37th IEEE International Conference on Software Engineering*, 2015.

[32] H. Christensen and K. Hansen, "An empirical investigation of architectural prototyping," *The Journal of Systems and Software,* vol. 83, pp. 133-142, 2010.

[33] M. Galster, S. Angelov, M. Meesters and P. Diebold, "A Multiple Case Study on the Architect's Role in Scrum," 2016.