TOWARDS ADAPTING NEXUS FOR PRACTICAL USE TO ELUCIDATE INTER-TEAM DEPENDENCIES: A LARGE-SCALE AGILE CASE STUDY

Lauren Christopher

A thesis submitted in partial fulfilment of the requirements for the degree

MASTERS (INDUSTRIAL ENGINEERING)

at the

FACULTY OF ENGINEERING, BUILT ENVIRONMENT, AND INFORMATION TECHNOLOGY

UNIVERSITY OF PRETORIA

SUPERVISOR: Prof. M de Vries

August 2022

ABSTRACT

Agile methodologies were originally created for small, collated teams; but large enterprises saw and wanted the benefits of agile for their projects that consisted of multiple developers, who may even be globally distributed. This raised the need for scaled agile and the issue of how to implement a small-scale solution on a much larger scale. Bringing together the empirical data found, along with experiences and information from practitioners who interact with these frameworks daily, an evaluation was done as to what solution would the most ideal for a Fintech company, comparing SAFe, LeSS, DAD, and Nexus as possible options. Through case study research, interviews and a survey, the issues of inter-team communication, collaboration, and dependencies were raised within the enterprise's banking build. Nexus was selected as the best large-scale agile framework for the enterprise.

This case study uses a Fintech company, building a digital banking platform, and in need of a large-scale agile solution, as they had globally distributed development teams working on the same project. Inspecting the banking build just after it's MVP (minimum viable product) launch, the learnings of Nexus, it's implementation and running within the enterprise, were used to understand how Nexus increased collaboration by daily communication which resolved dependencies between teams quicker by highlighting and focussing on inter-team dependencies in the Nexus Daily Scrum using a Kanban board which contained only these dependencies. Using these learning and comparing them to the theoretical version of Nexus, the study presented conclusions towards the adaptation of Nexus in a practical environment in order to better resolve inter-team dependencies. The case study provides knowledge and learnings surrounding Nexus and for those looking to implement a scaled agile solution. The enterprise, due to various factors, implemented only some of the Nexus components which leads to discussions on applications for future studies, as well as non-Nexus specific learnings that can be used as possible research topics.

Keywords: Scaling Scrum, Scrum, Scaling Agile, Scaled Agile Framework, Disciplined Agile Delivery, Nexus, Large-Scale Scrum, Agile, Large-Scale Agile

PREFACE

As a business analyst who was provided the opportunity to be part of, not only building a bank but also being involved in the company's first attempt at a scaled agile solution, I saw an opportunity to learn from a personal perspective as well as utilise the enterprise's learnings from such a big build for others in a similar sector to use. This thesis aims to provide an academic contribution to literature surrounding scaled agile solutions, especially regarding the Nexus framework, where there is an academic literature gap.

I could not have done this without the permission of the enterprise who allowed me to use them as the case study and be part of the digital banking platform build. Thank you for allowing me to participate, observe and analyse the process. To those, both within and outside of the organisation, who allowed me to interview and survey them, I am grateful for your honest and insightful contributions to this study. A thank you to the University of Pretoria who accepted this as a topic and allowed me to continue my studies through their institution; with a special thanks to my supervisor Prof. Marné de Vries for the constant support, meetings, and advice as I endeavoured on this academic journey, without your patience and direction I would not have been able to complete this. To those who assisted me from a support perspective, from proof-reading and editing to general words of encouragement and motivation, I cannot begin to express my gratitude. It was your help, your words, your advice that kept me going when things were overwhelming and for that I cannot thank you enough; with a particular mention to my parents who have never wavered in their support of me in general, but especially in my ability to complete this research.

The dissertation applies active voice, rather than passive voice, as advised by Hofstee (2006) in his book, titled: Constructing a good dissertation. The dissertation consists of two main parts, a synthesised report, consisting of about 100 pages, and multiple appendices that produce evidence of multiple data-gathering means.

I would also like to highlight that the following article has already been published in a peer-reviewed journal:

Christopher, L. A., & De Vries, M. (2020). Selecting a scaled agile approach for a Fin-Tech company. South African Journal of Industrial Engineering, 31(3), 196-208. doi:http://dx.doi.org/10.7166/31-3-2432.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1. Introduction

The purpose of this section is to provide the reader an overview of the problem that led to this study being conducted, giving background and context to the problem, as well as the problem statement. Following that, is a subsection on the research questions used to investigate and discover a possible solution for the problem. The thesis statement then follows with the parameters and more details of the study, specifically: limitations, definitions, assumptions and the significance of the study. Thereafter is a subsection on the publication potential of this research study, followed by the problem instance and solution validation within the enterprise. The last subsection will contain evidence of the need for scaling agile within the Fintech Enterprise (hereafter referred to as FE) environment regarding its digital banking platform build. Any interview transcripts quoted in this section can be found in Appendix A and the questions and answers to the surveys are located in Appendix B. Within the quotes, any non-italicised font indicates the researcher asking a probing question and italicised font in square brackets reflects having masked enterprise-specific naming or employees.

1.1 Problem context

Many companies have some form of software development department, or at least work with one. Due to the ever-increasing demand in the modern world for constant connectivity, software development is very quickly becoming a necessity for any business in today's market (Denning, 2015). The software industry is ever-changing and rapidly evolving; conventional methods for managing the development process, such as Waterfall, are diminishing (Hanssen, 2011) due to their long-term planning and structure which is usually rendered useless as soon as a project begins because the market has already changed so much from the time of planning to the time of development (Williams et al., 2003). As stated in van Wessel et al. (2021), most software development which runs using agile works better that then those using Waterfall, in time, budget, and user satisfaction. Another issue with these traditional methods was that they were based on a strong sense of hierarchy which, due to its nature, slowed down decision-making because of the bureaucracy associated with the number of approvals which needed to be made until a decision was cleared (Denning, 2015). The ideology to combat these issues was agile, which has been present in the workplace since the 1990's (Ebert et al., 2017; Williams et al., 2003); its quick adaptability, with the 'if you're going to fail, then fail fast' approach, is the reason it became so popular within the software development realm despite the fact that it originated from within the Toyota manufacturing plant (Denning, 2015).

Agile is a manifesto, a guideline, with general values and factors to follow. The basic themes and characteristics linked to agile are: 1) the interaction between team members and/or with the

customers should be interactive and 'face-to-face' rather than using tools and technology; 2) emphasis is placed on development and actual product over the need for extensive documentation; 3) creation of self-organising teams so that the strengths of its members are utilised; and 4) adaptability to change via an iterative process of constant review, change, and planning (Denning, 2015; Williams et al., 2003). Ebert & Paasivaara (2017) maintain that agile has definitely changed since its inception, initially it was created around ideologies such as "customer onboard" and "software before documentation"; but with more practical case studies it was realised that other business factors, such as governance, would also need to be considered, and not just software development, as they all were inter-linked and affected one another. It has however been the base for numerous frameworks/lightweight methodologies to evolve, such as Scrum, Kanban, Extreme Programming (XP) and Crystal, to name a few (Denning, 2015; Edison et al., 2021; Reifer et al., 2003; Uludağ et al., 2021; van Wessel et al., 2021).

The ideal scenario to run an agile project was when the teams were small, self-organising, collocated, and customers which were on-site; this would mean the teams could always engage in face-to-face communication and therefore any issues could be communicated and eradicated almost immediately (Reifer et al., 2003; Uludağ et al., 2021). In 2002 the question of scalability within agile was raised and became a topic of focus (Williams et al., 2003). In a consistently changing, technologically-driven modern world, which has a high demand on software and thus its development; the reality is that there are large teams which are globally distributed working together on major projects, who need to utilise agile methodologies in order to manage their software development in the best possible manner (Ebert et al., 2017). Whether one agreed that agile should only be used in its original context or not; it was agreed that it would still be used practically in a scaled form, and therefore guidelines to accommodate this were a necessity (Reifer et al., 2003). This gave rise to various scaling agile frameworks, some noted within industry currently are Scrum of Scrums (SoS), Scaled Agile Framework (SAFe), Large-Scale Scrum (LeSS), Disciplines Agile Delivery (DAD), Lean Scalable Agility for Engineering (LeanSAFE), and Nexus (Denning, 2017; Edison et al., 2021; Prikladnicki et al., 2016; Uludağ et al., 2021; van Wessel et al., 2021). The rise of the need for agile to be used in bigger projects, throughout an enterprise has been given the title 'large-scale agile development', which means to use agile for a project with multiple teams, which have 50 team members or a minimum of 6 teams (Uludağ et al., 2021). Scaled agile frameworks came to be in order to combat the complexity of a large scale project, enabling better delivery on business value as well as increasing the customer connection, better adaptability and ability to deal with the fast-changing environment, and to improve aspects of working in a large team such as collaboration, coordination and communication among teams while managing inter-team dependencies (Uludağ et al., 2021).

This study was initiated by the author who was working in a company within the financial sector at the time; it was first and foremost a financial services company but was largely driven by technology with a substantial in-house development team. As a business analyst, the author was involved in the company's biggest product development within recent years. The company in question, the FE, is a division of a larger group which is an SME (Small to Medium Enterprise) founded in 2015. At the time of this study, the FE was launching a new financial product, their digital banking platform, and was having to evaluate the way in which the company, at the time, had implemented Scrum. This was a vast project and, on such a large scale, with so many development teams involved, there was a need to look at the scalability of its current Scrum framework. The banking build, using an agile framework, is considered a large-scale agile development as this project did have over 50 members within its software development teams. This fits the definition of 'large-scale agile development' as stated in the previous paragraph but also used in Moe et al. (2021), who states the definition by Dikert et al. (2016, p. 50) as, "*software development organizations with 50 or more people or at least six teams. All people do not need to be developers but must belong to the same software development organi[s]ation developing a common product or project, and thus have a need to collaborate*."

The context of this study is that of scaling Scrum, using the FE's environment and specifically its digital banking platform project, as a 'real-world' case study. The scope of this study will focus on the challenges of adapting Scrum to suit a much larger group with various globally-distributed teams involved, using literature-based scenarios and results, as well as that of experts in the field. The case study highlights factors that were considered when selecting a scaling agile framework, and how these contribute to elevating the issues within the FE's digital banking platform build, as well as how the enterprise had to change the scaled agile framework selected in order to suit their environment.

1.2 Problem statement

The problem is that agile was initially created to be applied on a small-scale with a limited number of individuals and, when agile is scaled up, there are inter-team elements which were not catered for in the original agile methodology.

Most software development teams involved in large-scale projects are typically greater than the originally-specified small group of developers (fewer than 20); but the benefits of agile make it a perfect solution for software development, being able to adapt to a rapidly ever-changing environment and allowing for research and design in a quick, learn-and-adapt fast manner. With

numerous teams, potentially not collocated, all working on a single project, crucial elements such as communication and dependencies exist, and need to be addressed and managed.

1.3 Research questions

In this section the primary research question, along with seven secondary research questions, are stated.

1.3.1 Primary research question

How can the theoretical Nexus framework be adapted in order to make it a *practically viable solution* to improve knowledge on inter-team dependencies within a Fintech enterprise?

This study will use the metrics, i.e. *key performance areas* (KPAs) to evaluate what determines *practical viability* to improve knowledge on inter-team dependencies. In this study, *inter-team dependencies* will be evaluated as the main area of focus, whereas *communication* and *collaboration* are the KPAs that are assessed. The main objective is to find out what adaptations contributed towards better communication and collaboration that are required due to inter-team dependencies. The study also acknowledges that other interventions and circumstances could have contributed towards the increase in inter-team dependencies, these will be considered in the section 3.4, rival theories and risk mitigation strategies.

1.3.2 Secondary research questions

The research method used was case study research and therefore used Yin's guidance (Yin, 2018), as well as a need to understand the gap in the practical and theoretical Nexus application.

Question 1: What issues were experienced at the enterprise with regards to their use of Scrum?

Question 2: Who/what is most affected by the identified issues?

Question 3: Does the problem instance feature as a class-of-problems in existing literature?

Question 4: What knowledge areas could be useful to address the class-of-problems?

Question 5: What was learnt, in literature, from the implementation of scaled agile frameworks?

Question 6: What are the key components which can be compared to evaluate the differences in the practical and theoretical frameworks?

Question 7: What were the reasons that the theoretical key components did or did not work within this enterprise's environment?

1.4 Thesis statement

The theoretical Nexus framework needs adaptation to make it a *practically viable* solution to improve knowledge on inter-team dependencies within a Fintech enterprise.

1.5 Scope demarcation and limitations

The scope demarcation and limitations of this dissertation are as follows:

- This study was done within the context of a financial technology company. It can therefore not be stated that the findings will hold true for other business sectors.
- This study was done within the context of a banking digital platform build. It can therefore not be stated that the findings will hold true for other digital platform builds.
- This study was done within the context of a medium-sized enterprise. It can therefore not be stated that the findings will hold true for other sized enterprises (small or large).
- This study was done within the context of an enterprise which already had Scrum implemented. It can therefore not be stated that the findings will hold true for enterprises which do not have an established Scrum environment already implemented.
- This study was done within the context of globally distributed agile teams. It can therefore not be stated that the findings will hold true for collocated agile teams.
- Due to lack of empirical data which would limit this paper to a very small pool of research, non-empirical data was used to aid in getting a broader and therefore more inclusive range of research. This is further explained in section 2.1 (protocol for data synthesis).
- This paper focuses on scaled agile solutions and their relation to inter-team collaboration.

1.6 Definition of terms

BA: Business Analyst. This is an employee within the FE who gathers business requirements, turns them into specifications and assists with development build management, including testing and deployment assistance.

CLD: Causal-loop-diagram. “A typical CLD consists of a set of symbols representing a dynamic system’s causal structure: variables, causal links with a polarity and symbols that identify feedback loops with their polarity” (Schaffernicht, 2010, p. 653).

DAD: Disciplined Agile Delivery. “Disciplined Agile Delivery (DAD) is a people-first, learning-oriented hybrid agile approach to IT solution delivery” (Agile, n.d.).

FE: Fintech Enterprise (FE) is the masked name given to the financial technology (also called Fintech) enterprise being used as a case study in this dissertation.

LeSS: Large Scale Scrum. "Large-Scale Scrum (LeSS) isn't new and improved Scrum. And it's not Scrum at the bottom for each team, and something different layered on top. Rather, it's about figuring out how to apply the principles, purpose, elements, and elegance of Scrum in a large-scale context, as simply as possible" (Company, 2019).

LTM: Lead Technical Manager. This was the lead on all the highly technical items within the digital banking build.

Nexus: A scaled Scrum framework. "Nexus is a framework for developing and sustaining scaled product and software delivery initiatives. It uses Scrum as its building block" (Schwaber, 2018, p. 2)

PDL: Product Development Lead. The lead of the digital banking platform build project.

PO: Product Owner. This a Scrum role, and it the person responsible for managing the product. In the FE, the PDL and the PO were the same person.

RCA: Root-cause analysis. "Root cause analysis is a systematic process used to address problems or non-conformance to identify the source of the problem" (Connelly, 2012, p. 316).

SAFe: Scaled Agile Framework. "SAFe® for Lean Enterprises is a knowledge base of proven, integrated principles, practices, and competencies for Lean, Agile, and DevOps" (Agile, 2018).

SLR: Systematic Literature Review. "A rigorous stand-alone literature review, according to Fink's (2005) definition, must be systematic in following a methodological approach, explicit in explaining the procedures by which it was conducted, comprehensive in its scope of including all relevant material, and hence reproducible by others who would follow the same approach in reviewing the topic" (Okoli et al., 2010, p. 1).

SM: Scrum Master. This is a Scrum role, and SMs are responsible for implementing and ensuring the running of the Agile solution within the enterprise.

SoS: Scrum of Scrums. "A fairly universal practice for coordinating work among several teams is the scrum of scrums meeting" (Cohn, 2010, p. 340).

1.7 Assumptions

Assumptions are points which are considered true by the author and therefore are to also be considered true by the readers, whereas limitations are factors within the study that the author/researcher cannot control. The following assumptions are to be considered with this study:

- It is assumed that the answers provided by those interviewed and surveyed are completely honest.
- It is assumed that the reader has some background knowledge and understanding of agile and its frameworks.

1.8 Significance

Agile, with its benefits of reduced lead time and higher workplace efficiency within teams, has become a very popular methodology for software development in current times. The need for a scaled agile solution has therefore become an ever-more relevant topic as large enterprises with big projects hope to reap these benefits, despite the fact that agile was originally created for projects on a much smaller scale (Paasivaara et al., 2013). With big projects, some challenges exist, such as inter-team communication and dependencies, as further discussed in Chapter 2. These challenges can become a burden to a company if they are not managed and could cause the project to fail and cost the enterprise severely, hence, they need to be taken seriously and mitigated as far as possible.

The theoretical significance of this topic is that scaled agile frameworks have been well documented, including reports of their industry stories and case studies, however there is a considerable gap in research regarding their actual implementation, results and best-suited environments (Dikert et al., 2016; Ebert et al., 2017; Paasivaara, 2017). There is a lack of empirical data and research within the scaled agile field, this has been mentioned numerous times, focussing on the need for more supported knowledge with regards to the scaled framework's transformation, results, challenges, success factors, their evolution, and effectiveness (Conboy et al., 2019; Dikert et al., 2016; Ebert et al., 2017; Francino, n.d; Paasivaara, 2017; Paasivaara et al., 2014; Paasivaara et al., 2013; Uludağ et al., 2021). As stated very clearly in Paasivaara (2017, p. 36), *"[a] recent systematic literature review revealed the lack of systematic studies on large software development organi[s]ations adopting agile methods. The review found only six scientific studies on large-scale agile transformations, while almost 90% of the included papers were experience reports"* and even more recently in Conboy & Carroll (2019, p. 1) , *"However, empirical evidence regarding the adoption of such frameworks, their use, effectiveness, and challenges is still very much in its infancy."* Without proper research and empirical data to support and analyse the frameworks, how does an enterprise make an educated decision on which, if any, are best suited to their environment? The incorrect decision with regards to the implementation of something as critical as a framework, which will affect the way an enterprise is run, can be catastrophic.

With the aforementioned paragraphs and Chapter 2, which elaborates on the areas of concern, as well as the suggested solutions within literature, there is significance of this particular study within the theoretical realm, as well as the practical. The case study presented in this dissertation provides new insight on how a theoretical scaled agile approach, Nexus, had to be adapted to make it *practically viable*, addressing a key concern that for large-scale projects, namely a lack of knowledge regarding existing or required inter-team dependencies.

1.9 Problem-instance and solution-need validation

In this section, the researcher provides a thorough investigation into some of FE's problems, before the FE experimented with a solution, answering two of the secondary research questions of this study, i.e.: *Question 1: What issues were experienced at the enterprise with regards to their use of Scrum,* as well as *Question 2: Who/what is most affected by the identified issues?*

Once validated as a problem instance experienced at the FE, it is also investigated whether the problem instance exists as a larger class-of-problems in literature, in section 2.2, validating the significance of the research. In this section, there is also validation that multiple stakeholders at the FE believed that a solution was required. Once a solution-need was validated, there was a search for the existing body of knowledge of solutions that may be appropriate for the FE, reporting on these solutions in section 2.3.

Any enterprise which utilises agile within their environment, will need to look at a scaled agile framework for any projects which involve multiple teams or large groups of developers. This scenario is more than likely with any large-scale build or development, no matter the organisation. With the ever-increasing need to deliver more and get to market sooner, using a scaled agile approach becomes mandatory for many enterprises. *"There are many examples of where people have used scaled Scrum, Microsoft uses it to work with like 2500 teams across the globe at once. Google uses some forms of scaled Scrum, Cathay Pacific uses Nexus. The FBI implemented scaled scrum as well…"* as stated by one of the research participants, referred to as Scrum Master 2 (SM2) for the purpose of this study/paper. Also, confirmed by the research participant External Scrum Master, *"[t]here are many instances where the size of the projects being initiated are too large to be done in an ad hoc, uncoordinated manner. Scaling agile for these projects becomes an imperative"* The FE is a company that utilises Scrum in their environment, and the development of its digital banking platform being its biggest build in the few years, a scaled agile framework was certainly required. In section 1.9.2 there is evidence that, within the enterprise, the current Scrum solution in place was not the correct framework for the bank build project, and in section 1.9.3 the need by stakeholders for a solution to this issue is identified.

1.9.1 Data-gathering and analysis strategy and diagnosis techniques

The study applied a number of strategies for data-gathering, the first was *interviewing*, the second *observing*, and the third a *short survey*. Since structured interviews were used, a *structured codebook* was useful in analysing interview data in a structured way. Motivations for the selected data-gathering and analysis instruments follow the subsequent paragraphs once the need was elaborated on in order to *diagnose* the root causes of an existing problem.

Diagnosis techniques are not well established with regards to agile frameworks, i.e. there is no set or commonly used method of analysing the agility of a framework or evaluating the performance of the framework (Gill et al., 2006). For this reason the author suggested two generic problem diagnosis techniques, a root-cause-analysis (RCA) utilising the Five Whys method to identify the root causes of existing problems, and a causal-loop-diagram (CLD) which is used as a visual aid to the RCA, in order to show the influences that the causal variables have on each other and how they are connected (Spector et al., 2001).

An RCA is a means to closely identify, analyse, and understand the underlying reason for the issues presented (Connelly, 2012). RCAs identify, define and understand the problem, leading to the identification of the root cause and the subsequent creation, testing, and monitoring of a solution in order to resolve the root cause (Bresky, 2007). There are various methods used within an RCA to determine the root cause, such as cause and effect charts, matrix diagrams, Five Whys, and fault tree analysis (Bresky, 2007). The *Five Whys* was the method selected for this task as it was a quick and efficient way of evaluating the FE's current methodology and identifying the root causes. A causal-loop-diagram, also occasionally referred to as a feedback-loop diagram, is a means of showing either the positive or negative affects certain variables have on each other, in a visual manner (Spector et al., 2001). CLDs utilise arrows, known as causal links, to show both the link itself and direction of the link. Each link is also accompanied with a symbol of polarity, showing if the relationship is either negative or positive , if the variable being examined is being increased or decreased (Schaffernicht, 2010).

The *observation* data-gathering strategy was employed because the author was not only working within the environment, but also on the project, full-time at the time of this study, and was therefore in a position to use participant observation. Participant observation is a useful data-gathering strategy as is allows for the observer to notice nonverbal cues, such as emotions and feelings towards a situation, which may not be communicated within an interview because the interviewee is choosing to be polite/politically correct in order not to cause conflict (Kawulich, 2005). It also serves as a means of verification and deeper understanding within the interviews as the observer can relate and critically assess what is being said, as they have possibly experienced it and have a better sense of the environment.

The *interview* data-gathering strategy was employed in an attempt to better understand points-of-view from both the business and development teams' perspective. Interviewing is a form of qualitative data collection, which follows the general process of an initial pre-interview contact in order to ask and inform the interviewee of their interview, drafting the questions, conducting the interview with clear communication, and then transcribing the interview from whatever device it was recorded on (Byrne, 2001). The interviews were one-on-one between the author and the interviewee, in a closed room for complete privacy. Interviews are useful in understanding the interviewees' personal opinions and beliefs on topics, as well as their experiences within the environment with which they are involved (Harrell et al., 2009).

Due to this build being the FE's largest in recent years, several people were interviewed. The two main employees in charge of the project were initially chosen to be interviewed. The first interviewee was the Product Development Lead (PDL) from the business' perspective; second, the Lead Technical Manager (LTM), who is heading the project from a technical perspective. These two individuals were responsible for making all key decisions within this project. Interviewing the PDL and the LTM, both of whom understood the project in its entirety and were themselves working on the project, were thought to be the best approach to obtain a reliable, quality overview and perspective from each of the two main sides of this project, business and development. The senior business analyst working on the project, Senior Business Analyst (SBA), was also interviewed as she was instrumental in connecting business, and their requirements, with development through the design of systems and processes to ensure that the business requirements were met and could be built and understood by the developers. The last two people interviewed were the two Scrum Masters (SMs) involved in this project since its inception, namely Scrum Master 1 (SM1) and Scrum Master 2 (SM2), who implemented, maintained and championed Scrum with the company overall, and would be doing so at a scaled level within the bank build as well. The bank build product was the main driver for FE's decision to investigate their existing Scrum methodology and scaling it in order to deal with the magnitude of this project.

In addition to the above interviews, a *survey* was sent to an External Scrum Master, who is impartial to the business environment in question and purely agile knowledge driven. A survey is a means to ask a fixed list of questions, to all those who take it, providing a standardised method of collecting data (Harrell et al., 2009). The survey was e-mailed to the participant, allowing him to take his own time and answer the questions as honestly as he would like, without the added pressure of time or somebody sitting and listening. Another reason for asking the External Scrum Master was their personal experience in DAD and scaled agile framework implementations, with the ability to give an expert opinion on the agile and scaling agile topics.

The questions were kept to a minimum and were asked in a straight-forward manner so as to not to confuse the participant, as nobody would be sitting with them to provide further clarification on any of the questions should they require it.

Table 1 provides the details of the persons who actively participated in the data-gathering and diagnosis technique, along with what type of data-gathering they participated in.

Table 1: Persons who participated in the data-gathering and diagnosis techniques

Person	**Participated In**
Lead Technical Manager	Interview
Product Development Lead	Interview
Senior Business Analyst	Interview
Scrum Master 1	Interview
Scrum Master 2	Interview
External Scrum Master	Survey

A *structural codebook* was used with the interview data, based on Guest et al.'s Chapter 3 (Guest et al., 2012), included in Appendix C. It was used to cover themes such as *area of concern*, including the causes, as well as *suggested solutions*.

- The first interview question was aimed at the SMs, specifically in order to gain a better understanding of the background to agile and Scrum.
- In the second question, the interviewees were asked to describe, in their opinion, the difference between was they deem an ideal work environment should be versus what their current working environment is. The answer to this question was used to illustrate a gap between the two environments and possibly trigger the start of any concerns that there may be, as well as lead into the next set of questions about the current working environment.
- Questions three to six were intended to uncover what was working and what was not working within the current setup of the enterprise structure currently. Analysis of the current environment provided beneficial insight into what the areas and causes of concern.
- The seventh question was designed to bridge the areas of concern and that of the suggested solutions, by enquiring directly with the stakeholders as to their need for a solution. The

answer to this question was the lead for the next grouping of questions, questions eight to ten, which focused on the solution of scaled Scrum.

- Question 10 was directed at the SMs in order to provide examples of scaled Scrum as support to the viability of the proposed solution by making reference to examples in larger, more complex, organisations.

1.9.2 Results for validating problem instance

At the time of the study, the company had already implemented Scrum, as stated by the PDL, *"we were using Scrum, but in isolated manners."* This framework was working in some respects, as stated by SM 1, *"I think, in pockets, we found that some of, most of, the teams had really good cadence; proved to deliver properly and well within Sprints; team morale was up; people were working well together; people were understanding strengths and weaknesses of each other; so the teams actually were working in a very happy way."* This was considered excellent at the time as it had been a notable change from the initial waterfall framework and allowed the company to move forward, as explained by SM2, *"[w]hat Scrum allowed us to do was accept that we work in a modern world, a different world to what Waterfall solved…"* but it too had its disadvantages; such as lack of focus as stated by SBA, *"..and I feel like the focus sometimes fizzles a little bit."* Due to Scrum being so heavily focused within the teams, factors outside those teams were not considered, which gave rise to bottlenecks not being addressed. This issue, according to SM2, was resolved but led to the need for a leadership board. SM2 states, *"[t]he leadership board was an overarching board that was placed just before the product backlog where all our initiatives for the next couple of years would be listed and recorded. This gave us an additional redundant step because we were just creating silos again."*

Using the interview transcripts, the *Five Whys* method was executed in order to identify the root cause. Figure 1 shows a diagram that provides a quick visual representation of this method and its results. The Five Whys originated with the main concern of the inter-team dependencies being a weakness in this build, as mentioned by the LTM, *"…the inter-team dependency though was where things started to break down a little bit."* The first 'why' addressed the lack of communication between the teams, as stated by the PDL (also mentioned in the following section), *"We had many teams, all working in isolation, there were dependencies that they had amongst each other; but those dependencies weren't clearly communicated and understood, as a result impacted their own deliverables."* Teams weren't communicating, therefore how could inter-team dependencies be known by the those who needed to build/assist with the dependencies? The lack of communication problem was investigated with a second 'why', which put forward the lack of collaboration, also raised in the aforementioned quote by the PDL.

With the teams working in isolation there would be very little to no collaboration, resulting in the teams working in silos, as raised by SM2 in the previous paragraph; this resulted in the third 'why', which was teams working in silos. The fourth 'why', Scrum implemented within individual teams, brought clarity and reasoning to the isolation and lack of collaboration. The Scrum implementation within the company was executed at an individual team level; as mentioned by the PDL in the previous paragraph and indicated by SBA, *"[w]e had Scrum within our small groups, that's about it."* This brought the RCA to its final 'Why', which identified the root cause to be the fact that the Scrum implementation within the current system was not a system-based implementation but rather that of a team-based foundation; validated by SM2, *"[w]hat didn't work was that we realised that Scrum, although many people think it's a team-based thing, it's actually a systems-based thing. If you fixed the team, or you improve them with Scrum, you're going to hit a glass ceiling at some point because you can only optimise them so much before the organisational gravity pulls them back into bad habits or impacts their productivity. So, we optimised the team and once the team reached a certain level of productivity, we realised that factors outside of the team started impacting the team so that they couldn't grow more. They reached an optimum productivity level and then it kind of dipped because there wasn't any motivation. That was one of the big things we had to fix. Scrum wasn't only a localised team thing; it was a much bigger problem. If you wanted to fully tap into the power and benefits of Scrum, you needed to look at the system as well, to look at bottlenecks before and after your function."* Scrum, or any agile framework, should be implemented on a system and organisational level and not on a team level as it was, confirmed by External Scrum Master, "*If teams adopt agile as a framework without the understanding and support of the organisation, structures and processes within the organisation will not change to support the implementation, and the initiative will struggle to survive – teams cannot work in isolation, it must be a system wide adoption."*

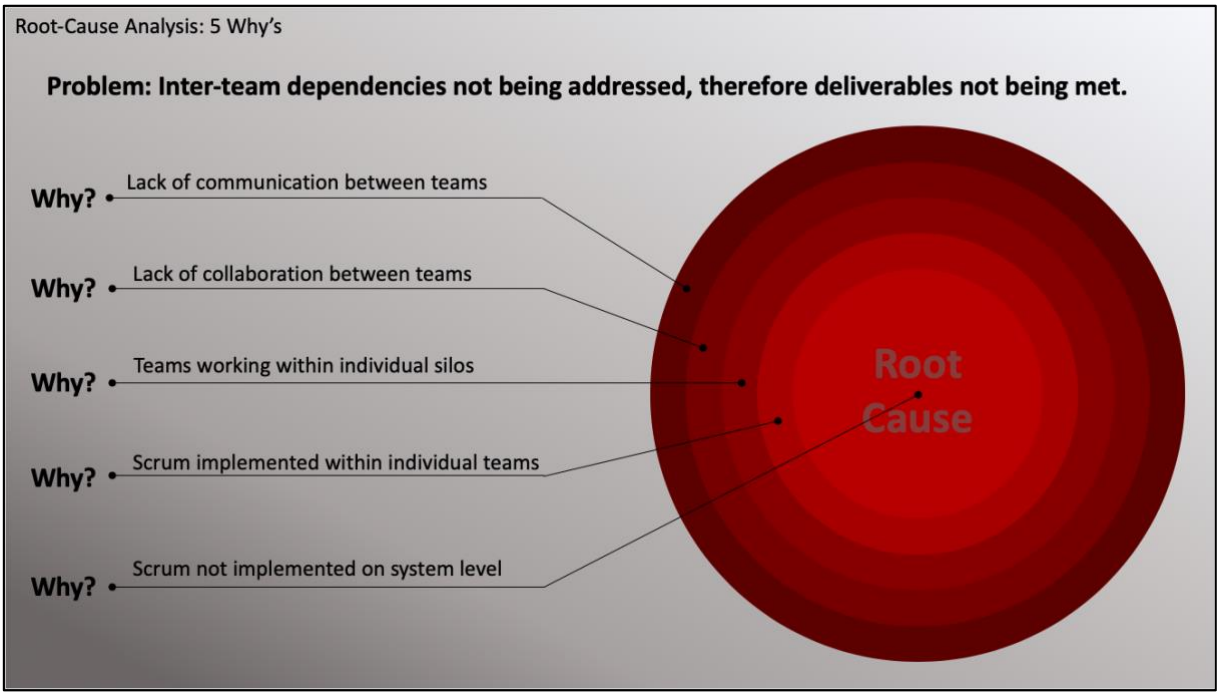


Figure 1: 'Five Whys' Results Represented Visually

Using the causal-loop-diagram found in Figure 2, three variables are causally linked, i.e. the knowledge of the inter-team dependencies, collaboration, and communication amongst the teams. The variables are all impacted positively, indicating a positive feedback loop, making this a reinforcing loop. If the knowledge of the inter-team dependencies was plotted against time, it would increase as time progressed, i.e. if the communication and collaboration increased, the knowledge of the inter-team dependencies would increase as well, therefore the issue of dependencies would no longer be a weak point in the build.

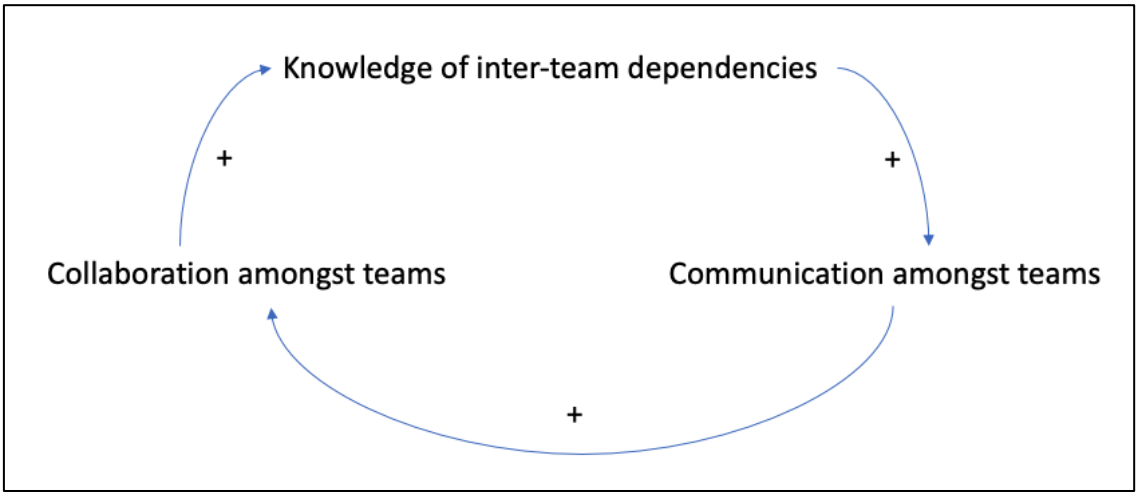


Figure 2: Causal-Loop-Diagram

1.9.3 Stakeholders or users that need a solution

Teams worked in silos due to the nature of the Scrum framework being implemented within the individual teams, *"[a]lso, it was nice because teams were going along nicely but we had no visibility of what everyone was doing, so this team would deliver something, but other teams didn't know what they were delivering"* as explained by SM1, so there was a lack of transparency between teams. The digital banking platform build however, was the biggest project the company has taken on in the recent years. Issues, such as the lack of clarity within the current environment, led to inter-team dependencies not being catered for, as raised by the PDL, *"we had many teams, all working in isolation, there were dependencies that they had amongst each other; but those dependencies weren't clearly communicated and understood, as a result impacted their own deliverables."* The LTM also touched on this issue, *"…the inter-team dependency though was where things started to break down a little bit. It was hard to coordinate between teams, especially when deliverables were due, and a team was dependent on two or three other teams. They might be able to work directly with one team, because we have two teams in India and then a couple teams in South Africa; if you were here and you were dependent on everyone here it was relatively easy to get things going. If you were dependent on teams in India, then you were locked to their dev cycle; they had already decided what they were going to do in their sprint before, they had no idea that you were looking for anything from them."*

Not only was the need for clear inter-team communication imperative, the teams themselves were globally distributed between India, South Africa and Mauritius (not mentioned in the LTM's comment, but there are two teams abroad); this added to the complexity of the project quite significantly. Another contributing factor was that there were only two SMs to facilitate all these teams, which was a comment both the LTM and SM1 raised respectively, *"[i]t was a bit of the global distribution and the teams had different Scrum Masters as well, so the Scrum Masters would get together and talk but they wouldn't be able to discuss the details of what the teams were doing, there would just be an overall 'this is where we are at' type of thing. Since they aren't involved in the actual building, the development work itself, they don't understand all the possible dependencies that could exist,"* and, *"[w]e were only two Scrum Masters and we realized that the magnitude of this product, having to logistically facilitate all the teams. So, what we needed to do was we needed to get, somehow, to a position where we could all work on one product and all the teams have visibility of each other. Within the agile world, having visibility of each other, and understanding what everyone is doing on one product is defined as scaled."*

The magnitude of so many teams working together, building one product, needed to be considered and catered for, as it was not a factor to which the company environment was accustomed. As stated by the PDL when asking **what caused the enterprise to consider a scaled solution**, *"[t]he size of the project," "and the many teams involved, there were 6 systems we needed to build,"* as well as *"...it [Scrum] made sense for what they were doing at that point in time. Small, minor, changes on systems, focused on them and executed them, not a lot of involvement with other people; versus what we do now, where its massive systems big overall, and complete redesign..."* With such an extensive project involving several globally distributed teams, transparency among the teams and clear communication were vital in addressing the critical issue regarding inter-team dependencies, identifying the distinct need for a solution within this project.

1.10 Document structure

In accordance with the phases of the process, the remaining parts of this dissertation are structured as follows:

Chapter 2 is the systematic literature review (SLR) which includes the research protocols used for the SLR, as well as what could be discovered in literature regarding the similar class-of-problems and any suggested solutions to assist with these. There is also a discussion to conclude the findings of the SLR.

Chapter 3 discusses the research methodology including literature and research on the methodology. The literature of rival theories is explored, as well as the practical application of rival theory analysis to use as a risk mitigation strategy. The ethical considerations of this study are also investigated at the end of this chapter.

Chapter 4 explains what Scrum is and goes into full detail of the theoretical Nexus framework. The chapter explains what Nexus is and how the framework is setup, listing and describing the various components which are used specifically within Nexus.

Chapter 5 is the demonstration of the Nexus theoretical concepts and practices that were used, versus those practices that deviated from the theory.

Chapter 6 presents the results of the case study, using the information extracted from interviews during the data gathering process, to understand *what was adapted* for the practical application of Nexus and the *reasoning* for these adaptations.

Chapter 7 is a synthesis of the demonstration and results, using the information from Chapter 6 to understand why the adaptations shown in Chapter 5 were needed, given reason and explanation for the difference in the theoretical and practical application of Nexus. Similarities

and differences, along with reasoning, between the theoretical and practical Nexus are synthesized in a table-format for ease of visibility and practicality. Any other learnings that emerged from this case study are also presented in this chapter.

Chapter 8 concludes the dissertation with a summary, where the research questions are re-visited and answered, as well as recommending future studies within this area of knowledge.

References lists all the reference used in this research paper.

Appendices A to I contain the appendices with interview transcripts, survey questions and answers, structural codebook, quality assessment scoring, the extracted data form, both data coding sets, the inter-coder agreement comparison, and the thematic analysis coding.

2. Systematic Literature Review

The following section presents the results of a systematic literature review, conducting a search for existing knowledge on a class-of-problems that relates to the FE problem instance, i.e. problems (such as communication and collaboration) when using an agile methodology within a scaled context, and solution types, i.e. scaled agile frameworks.

Since most researchers are required to use this research methodology if the intention is to publish the work, the literature review is regarded as one of the most used research methodologies (Okoli, 2015).

2.1 Research protocol for systematic literature review

To create the research protocol for the SLR, the 8-step procedure by Okoli (2015, 2010) was used, elaborating now on Step 1 (the purpose of the review) and Step 2 (the research protocol).

Step 1: The purpose of this literature review is to look at scaling agile frameworks with a focus on what factors/issues are considered when choosing such a framework, as well as the lessons learnt from other research scholars and practitioners who implemented them. As stated in section 1.3.2, some of the secondary research questions of the Masters study will be addressed using the SLR, including:

- *Question 3: Does the problem instance feature as a class-of-problems in existing literature?*
- *Question 4: What knowledge areas could be useful to address the class-of-problems?*
- *Question 5: What was learnt, in literature, from the implementation of scaled agile frameworks?*

Even though case study research was used for this paper, and for this research methodology the researcher primarily acts as an observer and not as a designer/developer of the solution, the need and development of the solution within the FE is the foundation of the case study's observations. Without this solution being implemented, there would be no case study to observe, and the background knowledge of the solution leads to a better understanding of why events were put in place, allowing for the investigation to provide a more in-depth analysis when reviewing final results. This level of knowledge also allows for improved future research by allowing researchers to draw comparisons in their cases to this particular study, and to take from this dissertation what would be most useful to their case.

Step 2: The research protocols defined for the systematic literature review are discussed hereunder.

Protocol for phases

This study applied Okoli's (2015, 2010) suggested phases and guidelines for conducting an SLR, i.e. planning, selection, extraction, execution, planning the review, conducting the review, and reporting the review.

Protocol for key words

The following keywords were used to search for applicable research documentation: "Scaling Scrum", "Scrum", "Scaling Agile", "Scaled Agile Framework", "Disciplined Agile Delivery", "Nexus", "Large-Scale Scrum", "Agile", "Large-Scale Agile".

Protocol of sources

The University of Pretoria's online library (http://www.library.up.ac.za) was used as the main searching point for viable sources. Several books and journals, as well as the following main electronic databases (all accessed via the University of Pretoria's library system), were utilised:

- World Cat (https://univofpretoria.on.worldcat.org)
- IEEE Xplore (https://ieeexplore-ieee-org.uplib.idm.oclc.org)
- SpringerLink (https://link-springer-com.uplib.idm.oclc.org)

Protocol for practical screen

The following inclusion and exclusion criteria were applied:

- Inclusions: Peer-reviewed online articles written in English and English books which contained any of the key words within their title. The peer-review criterion was included only for the initial research, data which arose from backward snowballing any latter research was not held to this criterion. The explanation for this decision can be found later in this section under *Protocol for quality appraisal*. Only research taken from the year 2000 onwards was considered, this is due to the scalability of agile being a point of topic in 2002 (Williams et al., 2003); and therefore includes a few years prior to this topic for understanding on how it came about and thus the origins of the topic.
- Exclusions: Any non-English documentation and any research documentation which had no connection to the key words used for this SLR. Due to the large number of free results found, any source which needed a payment for access was excluded immediately. While doing to the initial search, any duplicate articles found were also excluded. With respect to the primary sources found, any non-peer reviewed online articles were also excluded.

Protocol for snowballing

From the articles that were selected, backwards snowballing was completed. According to their titles, keywords, and abstracts, as well as being in line with the language and year range from the inclusion and exclusion criteria, articles which related to the research questions and key words were selected for further consideration. Articles, blogs, and websites which were found as a result of the backward snowballing process, although not peer-reviewed, were considered. The reasoning for this was that if these sources had been used to create content for a peer-reviewed article, then they themselves should be considered reliable sources. This rationale was used to ensure a comprehensive but focused pool of research was considered and utilised.

Protocol for quality appraisal

The quality appraisal process was taken from Kitchenham et al. (2008), that utilised the evaluation process from the York University, Centre for Reviews and Dissemination (CDR) and the Database of Abstracts of Reviews of Effects (DARE) criteria. The quality assessment is based on four questions which can be answered with yes, partially, or no, giving a score for each answer as 1, 0.5, and 0 respectively. Any source that scored a total of lower than 2 was not considered for this study. Each source is rated with this scoring metric against all four questions, allowing for the quality of the sources to be compared to each other, and ensuring all sources used are of the same high calibre.

The questions were as follows, displaying below each is the definition of the score answer options (Kitchenham et al., 2008):

Qual-Question 1: Is the selected article clearly focused on agile and its frameworks?

1. Yes: Agile and frameworks linked to it are explicitly mentioned and referred to.
2. Partly: Agile is mentioned and there is possibly a vague reference to related frameworks.
3. No: There is no mention of agile or its frameworks in the article.

Qual-Question 2: Is the scalability of agile clearly addressed?

1. Yes: The need for scalability within agile environments is openly addressed.
2. Partly: The need for agile to be scaled is mentioned, but there is no focus on this aspect.
3. No: There is no reference to scaling agile at all.

Qual-Question 3: Is there reference to frameworks/approaches that can be used in order to utilise agile in a large-scale operation (means of scaling agile)?

1. Yes: Possibly frameworks/approaches for using agile in a large-scale operation are mentioned.

2. Partly: There is mention of a way in which one can use agile in large-scale operation but not directly linked to a framework/approach.
3. No: There is no mention of a framework/approach so that agile can be used in a large-scale operation.

Qual-Question 4: Is there empirical evidence that a suggested framework/approach (as a solution) sufficiently addresses some the problematic factors associated with agile?

1. Yes: There is evidence of frameworks/approaches for scaling agile and it is clearly referenced as to how they have addressed problematic factors associated with agile.
2. Partly: There is evidence of frameworks/approaches for scaling agile but with vague reference to the problematic factors associated with agile that they have addressed.
3. No: There is no evidence of what problematic factors associated with scaling agile, have been addressed.

The quality scoring of the sources can be found in Appendix D. A summary of the result is listed in Table 2 hereunder, where a full score for the *Quality Score is given*, i.e. 4, indicates that a source scored 1 for each of the four quality questions.

Table 2: Source Quality Appraisal: Score Summary

Quality Score	Number of Sources
0	0
1	0
2	4
3	3
3.5	2
4	24

Only sources with a score of 3 or higher were considered as sources for section 2.3 (SLR results for suggested solution areas), since any source that scored lower than 3 only scored in the first two questions, i.e. Qual-Question 1 and Qual-Question 2 (please see Appendix D); meaning it did not relate to frameworks/approaches that are considered as suggested solutions. The first two questions addressed agile and touched on the idea of scalability, hence sources that scored in these questions were used as data for section 2.2. (SLR results for class-of-problem) in order to identify problems related to agile and seek out the need for a solution with regards to

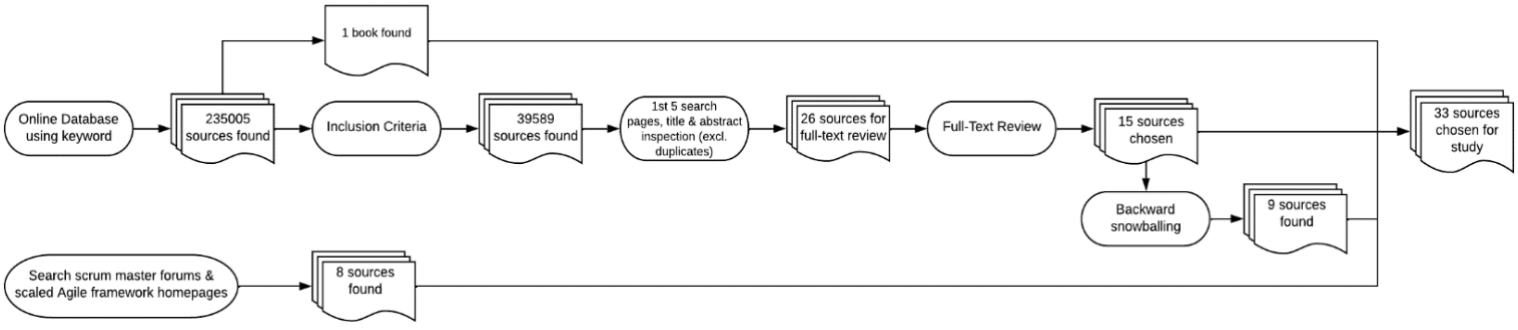
scalability. Sources which scored in Qual-Question 3 and Qual-Question 4 provided data relating to frameworks/approaches, hence suggested solution areas, which are appropriate for section 2.3.

Protocol for data extraction

For the primary/initial research, the search method used was to utilise the University of Pretoria's online library and search using the keywords (stated earlier in this section), 235005 results were found, from which one book was selected. The inclusion criteria (from the practical screen protocol) was then used to indicate date range and only peer-reviewed articles, this resulted in 39589 sources. Due to the high number of sources, the wide range of keywords, and there being only one analyst on this project; the researcher decided to go through the first five pages of results per keyword search (with the inclusion criteria) and select articles based on the title and abstract, excluding any duplicate matches found. The first search was done in 2019, ending in July of that year; with an additional, more recent search, being done in 2022 (ending in June 2022). This process led to 26 sources being selected for full-text review, after which 16 sources (including a book) were selected as primary sources. Backwards snowballing was then performed and based on their titles and abstracts, followed by a full-text review, a further nine sources were found.

There is not a lot of empirical data on scaled agile and its frameworks, in terms of comparisons of features or evaluation metrics/tools (Conboy et al., 2019; Dikert et al., 2016; Ebert et al., 2017; Paasivaara, 2017; Paasivaara et al., 2014; Paasivaara et al., 2013). The researcher decided to include sources of lower quality (blogs and web pages) to obtain better coverage of this information from practitioners and members of the agile community, including their perceptions of frameworks and their usefulness in their everyday careers. Using known Scrum Master forums, such as scrum.org and the scaled agile framework's homepages, guidebooks on frameworks, framework comparisons, case studies and possibly evaluation metrics were found in a further eight sources. With all the sources consolidated, a total of 33 sources were ultimately selected as the final sources for this study, as indicated in Figure 3.

Figure 3: Search method of number of sources found



A data extraction form was used to capture standard article information (author, title, year of publication, and publication type) as guided by Kitchenham (2004), as well as the extracted codebook themes to which the source was linked. The extraction form information can be found in Table 3 and the actual data extraction, using the form, is available in Appendix E.

The codebook was based on Chapter 3 of Guest et al. (2012) and can be found in Table 5. There was no second analyst available who was familiar with the coding process, therefore, the author adopted the intercoder-agreement guidelines from Chapter 4 of Guest et al. (2012) in order to perform an intercoder-agreement. The author created the codebook and then selected three sources, each of three pages in length (if the source was longer than three pages, only the first three pages were used), and explained the coding process and codebook to the other analyst. The author then created the *master set* of data coding (see Appendix F) and used the other analyst's code and created the *secondary set* of data coding (see Appendix G). These two sets were then compared with each other and the comparison scoring is available in Appendix H. The percentage agreement, calculated for the two sets of coding was 85%. An intercoder-agreement of 80% or higher is considered acceptable (Guest et al., 2012), therefore there was no adjustment made to the codebook. The *class-of-problems* was addressed in the extracted themes of 'OriginFW' and 'ScaleNeed', which focussed on any non-scaled agile solutions and the need for a scaled agile solution. The *solution* to the class-of-problems was addressed in the extracted themes of 'ScaledFW', 'ScaledFactors', and 'ScaledPros&Cons'; that provided information about scaled agile frameworks/approaches, any aspects which should be considered when scaling, and any positives and negatives of the scaled agile frameworks.

Table 3: Extraction Form Information

Source Identifier	A unique identifier given to each source, expressed as a #(number).
Author(s)	The author, or authors of the source content.
Title	The title of the data source.
Year of Publication	The year the source was published.
Publication Type	The type of publication the data source belongs to, for example nook, journal article, conference proceeding, blog, etc.
Peer-Reviewed?	Was the source peer-reviewed? This is a Boolean response, yes or no.
Topic	The main topic the source falls into, either non-scaled agile or scaled agile.
Extracted Themes	These are references to the extracted themes from the codebook, including 'AgileOriginal', 'OriginalFW', 'ScaleNeed', 'ScaledFW', 'ScaledFactors', 'ScaledPros&Cons', and 'ScaledExamples'.

Protocol for data synthesis

The data synthesis was conducted in a qualitative and quantitative manner, represented in graphical formats. These graphical representations assisted with comparing the amount and type of data which was collected regarding the class-of-problems and its solutions, making it easier to properly utilise and analyse the data.

The frequency of a theme in a source is taken as 1, regardless of how many times the theme occurs in the source. In other words, if the theme appears once or numerous times in a source, the theme's frequency is counted as '1' for that source in accordance with the guidelines provided by Guest et al. (2012). With this in mind, Table 4 shows the quantitative data, by mapping the years of publication against the extracted themes. Showing that even though scaling agile was recognised as early as 2003, scaling agile was only acknowledged from 2010, with scaling agile being raised as a top research topic in the 2010 XP conference (Paasivaara et al., 2014). The total number of sources per theme was 13, 18, 18, 23, 18, 20, and 14 for 'AgileOriginal', 'OriginalFW', 'ScaleNeed', 'ScaledFW', 'ScaledFactors', 'ScaledPros&Con', and 'ScaledExamples' respectively. For two of the selected sources, no representation is given in Table 4 since these sources were not dated and therefore cannot be used in Table 4. Figure 4 shows the percentage of data related to each extracted theme, showing that 40% of data falls within the non-scaled agile topic and 60% of the data falling within the scaled agile topic area, i.e. the largest percentage of data related to the general mentioning of various scaled frameworks. For the meaning of the extracted theme codes, refer to Table3.

Table 4: Number of sources based on the year of publication and extracted theme code

Extracted Theme	2003	2010	2012	2013	2014	2015	2016	2017	2018	2019	2021
AgileOriginal	1	0	1	1	1	1	2	5	0	1	2
OriginalFW	1	1	1	1	0	1	3	3	4	1	4
ScaleNeed	1	1	1	1	1	1	3	3	3	1	4
ScaledFW	0	1	1	1	0	0	3	6	7	2	5
ScaledFactors	1	1	1	1	1	0	2	6	3	1	4
ScaledPros&Con	1	1	1	1	1	1	1	5	5	1	3
ScaledExamples	0	1	1	1	0	1	1	5	2	2	2

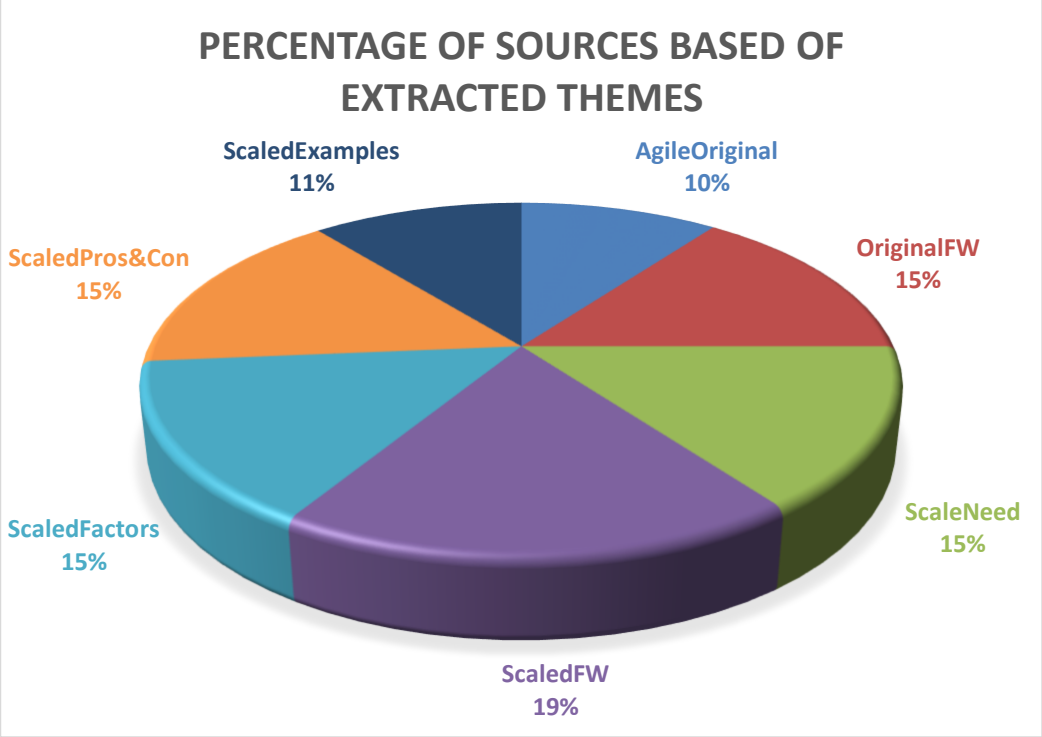


Figure 4: Percentage of sources based on extracted themes

Figure 5 is a graphical representation of data synthesis, showing the number and type of sources per extracted theme. The quality of sources has been grouped into *high*, *medium*, and *low*, based on the type of publication that was used. High-quality publications were considered to be any peer-reviewed articles/documents, books, or master theses; because these publications have been examined and reviewed academically, and have had extensive research as the foundation for their creation. Medium-quality sources were those which were non-peer-reviewed conference papers, guides, reports, case studies, and white papers. The non-peer reviewed sources were classified as medium-quality, since these publications may provide evidence/case studies supporting some claim contained within the high-quality sources, or they provide guidelines or a standard for implementation/use (in the case of guidelines). Lastly the low-quality sources are those which fall into the publication type categories of blogs or web pages, as they have not been validated/reviewed by other research scholars and have no academic evidence supporting the information. Since there is a lack of academic literature on the scaled agile frameworks, as explained and supported in section 1.8, the low-quality data was also included in the study.

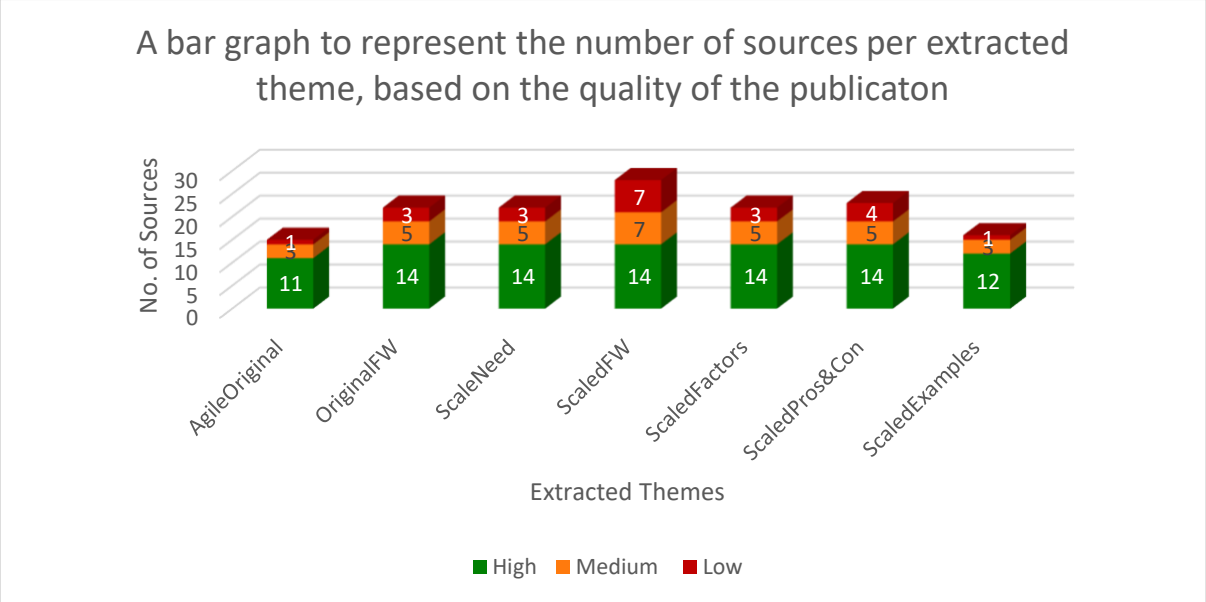


Figure 5: Quality of data sources per extracted theme

2.2 SLR results for class-of-problem

Using the codebook as cited by Guest et al. (2012) found in Table 5, the extracted themes such as ScaleNeed, ScaledFactors, and ScaledPros&Cons were used to show that there is a need for scaled agile solutions and factors as well as issues which need to be taken into account when scaling agile. The coding for these extracted themes (codes) is available in Appendix I. The tally of sources listed according to all the extracted themes can be found in the *protocol for data synthesis* in section 2.1. However, with reference to the extracted themes of ScaleNeed, ScaledFactors, and ScaledPros&Cons, the totals of sources for each are 22, 22 and 23 respectively.

Table 5: Thematic Analysis Codebook

Topic: Non-Scaled agile

Code: AgileOriginal

Brief Definition: Any information that helps the reader understand the origin of agile.

Full Description: Agile is an ideology which follows the agile manifesto, using its core principles to efficiently facilitate projects in industry. Agile is said to be a mindset, and its principles have been used in numerous frameworks.

When to Use: Apply this code to all references which relate to the history of agile, in which environments it is intended to be used, and the conditions in which it is was originally designed.

When Not to Use: Do not use this code for references related to frameworks/approaches which were used to implement and run agile principles in non-scale environments (see OriginFW).

Example: "agile was originally designed for a small, collocated team, working on one project."

Code: OriginFW

Brief Definition: Information on any of the non-scale agile frameworks.

Full Description: Agile's manifesto and principles have been used in non-scale agile frameworks such as Scrum, Lean and Kanban. They have applied the agile principles to create frameworks in order to assist enterprises with a means to make their enterprise agile.

When to Use: Apply this code to all references which relate to non-scale agile frameworks; including environmental/implementation needs, pros, cons, and learnings.

When Not to Use: Do not use this code for references related to general agile information which is not directly linked to specific frameworks.

Example: "Scrum was being used to …"

Code: ScaleNeed

Brief Definition: Reasons for the need to scale non-scaled agile systems.

Full Description: Agile was originally created to be used with small collocated teams but when the benefits of the methodology were realised, large-scale projects wanted the same benefits. Hence, for various reasons related to the enterprise environment, there was a need to find a way to scale agile for large-team projects.

When to Use: Apply this code to all references which relate to the need for a scaled agile solutions, this may include any factors which lead to this need and/or contribute to the need within the enterprise environment.

When Not to Use: Do not use this code for references related to issues within non-scale agile framework environments which are linked to problems that have no association to scaling agile.

Example: "The enterprise was using Scrum at the time but found the need to scale due to …"

Topic: Scaled agile

Code: ScaledFW

Brief Definition: When there is reference made to a scaled agile framework.

Full Description: As stated in AgileOriginal there is an ideology called agile which was created as a small-scale project solution but there was a need for it to be scaled for larger projects (as explained in ScaleNeed). This need led to scaled agile frameworks being created which catered for this large-scale project scenario.

When to Use: Apply this code to all references which relate to the background and creation of scaled agile frameworks, including environmental or implementation needs.

When Not to Use: Do not use this code for references related to pros and cons of any of the scaled agile frameworks (see ScaledPros&Cons), learnings from framework implementation (see ScaledFactors), or mention of industry where such a framework has been implemented (see ScaledExamples).

Example: "Nexus is a framework which scales Scrum and can be used in a …"

Code: ScaledFactors

Brief Definition: Aspects learnt from, and factors which affect, an industry implementation of a scaled agile solution.

Full Description: Lessons learnt from industry which should be taken into account when looking at scaling agile. These do not relate to any framework specifically, but rather to scaling agile in general, as well as what affects the framework's functioning either positively or negatively.

When to Use: Apply this code to all references which relate to elements which positively or negatively affect a scaled agile implementation or general lessons learnt from an industry implementation, include challenges and success factors of scaled transformation.

When Not to Use: Do not use this code for references related to the pros and cons of specific scaled agile frameworks (see ScaledPros&Cons), or information related to specific scaled agile frameworks (see ScaledFW).

Example: "While there are possible benefits, large-scale transformations have challenges such as …"

Code: ScaledPros&Cons

Brief Definition: This is any positive or negative result within an enterprise environment as a result of a specific scaled agile framework being implemented.

Full Description: When any change is made to an environment there will be results of the change, these results can be either positive or negative depending on the perspective from which they are being considered. Implementing a scaled agile framework within an enterprise environment will lead to both positive and negative outcomes.

When to Use: Apply this code to all references which relate to positive or negative outcomes due to the implementation of a specific scaled agile framework within an enterprise environment.

When Not to Use: Do not use this code for references which are not directly related to a specific scaled agile framework(s), including general points of positive and negative results due a scaled agile transformation (see ScaledFactors).

Example: "After implementing Nexus there were benefits such as reduces lead times and…"

Code: ScaledExamples

Brief Definition: Names of companies in industry where a scaled agile approach has been used.

Full Description: There are case studies and learnings that can originate from industry but knowledge of the actual company from where these learnings come is useful and gives credibility and some background to the data.

When to Use: Apply this code to all references which mention the actual names of the company at which the scaled agile framework was used.

When Not to Use: Do not use this code for references related to general industry learnings (see ScaledFactors) or pros and cons found within industry implementations of a scaled agile framework (see ScaledPros&Cons)

Example: "Apple and Autodesk use it, which shows scaling can be …"

Agile was originally created to be used by small groups of developers, who formed a team and were collocated whilst working on a project (Bass, 2016; Company, 2019; Kalenda et al., 2018; Paasivaara, 2017; Paasivaara et al., 2012; Paasivaara et al., 2013; Uludağ et al., 2021), but the benefits of agile, such as improved efficiency, were recognised and desired in larger enterprises (Paasivaara et al., 2013). The reality of software development is that large organisations usually have development teams which are larger than the originally intended agile team number, more than one team working on a project, as well as globally distributed teams (Company, 2019; Dikert et al., 2016; Ebert et al., 2017; Prikladnicki et al., 2016; Reifer et al., 2003; Schwaber, 2018; Uludağ et al., 2021); therefore a solution needed to be found in order to scale agile in larger organisations.

Besides having to find a solution to scale agile, there are issues when considering a scaled agile solution, such as management and larger team dynamics. Having a larger group to manage, leads to higher risk of management becoming overwhelmed with the amount of work to keep track of and losing visibility and control of the project (CollabNet et al., 2018; Scrum.org, 2018a). Working with a larger group of people and trying to manage, as well as control, the project, can lead to the paucity of requirements analysis and architectural planning (Fourie et al., 2017; Paasivaara et al., 2014; Paasivaara et al., 2012; Paasivaara et al., 2013). The larger team dynamic brings various challenges of its own; such as coordination among the teams, comprehension of how work fits into the larger environment, the increased resistance towards change, geographic distribution, pre-set culture, and consistency of coding standards; as well as required documentation, which goes against the agile principles but is needed with the

number of people involved including external parties (Bass, 2016; Cohn, 2010; CollabNet et al., 2018; Edison et al., 2021; Fourie et al., 2017; Moe et al., 2021; Paasivaara et al., 2014; Paasivaara et al., 2012; Paasivaara et al., 2013; Reifer et al., 2003; Uludağ et al., 2021; van Wessel et al., 2021).

2.3 SLR results for suggested solution areas

Using the codebook (as per Guest et al. (2012)) found in Table 5, extracted themes such as ScaledFW, ScaledFactors, ScaledPros&Cons, as well as ScaledExamples were used to indicate which solutions there are in literature for scaled agile solutions, including the learnings which should be taken into account when looking at scaled agile solutions. As stated in section 2.2, the coding for these extracted themes (codes) can be found in Appendix I, the information about the tally of sources per all the extracted themes can also be found in section 2.2. The tally of sources specific to the extracted themes ScaledFW, ScaledFactors, ScaledPros&Cons, and ScaledExamples, is 28, 22, 23, and 16 respectively.

There are various frameworks that have been released in recent years which are aimed at scaling agile, the prominent ones include: Scrum of Scrums (SoS), Scaled Agile Framework (SAFe), Disciplines Agile Delivery (DAD), Large Scale Scrum (LeSS), and Nexus (Agile, n.d.; Agile, 2018; Bass, 2016; Cohn, 2010; Coleman, 2018; CollabNet et al., 2018; Company, 2019; Conboy et al., 2019; Dikert et al., 2016; Dolman et al., 2017; Ebert et al., 2017; Francino, n.d; Kalenda et al., 2018; Paasivaara, 2017; Paasivaara et al., 2014; Paasivaara et al., 2012; Prikladnicki et al., 2016; QASymphony, 2018; Schwaber, 2018; Scrum.org, 2017, 2018a, 2018b; Verma, 2017). According to the 2021 15th Annual State of Agile Report (Digital.ai, 2021) regarding scaled Agile approaches; 37% of organisations are using SAFe, 9% SoS, 8% other, 3% DAD, 3% LeSS, Enterprise Scrum, Lean Management is at 2% each, Nexus at 3%, and 1% using Recipes for Agile Governance in the Enterprise.

SoS appears to be one of the original scaled agile approaches with it being mentioned in 2010, when scaled agile was only just being recognised as a needed research topic (Cohn, 2010). Every framework has its strengths and weaknesses, and even though SoS is one of the mainly used scaled agile solutions, there appears to be a significant issue with it. Due to the large number of people who attend the SoS meetings from the various teams involved, the meetings became too long and there was a lack of task overlap with each individual group's set of work, limiting interest and therefore usefulness for attendees (Bass, 2016; Edison et al., 2021; Paasivaara et al., 2014; Paasivaara et al., 2012). This was best affirmed in Paasivaara et al. (2012, p. 237) stating, *"SoS meetings seem to work poorly when they have too many participants with disjoint interests and concerns; and smaller, focussed inter-team meetings*

with participants having joint goals and interests, seem to have a better chance of being perceived as successful". SoS, being an earlier-scaled agile approach, would have been used as a base for the scaled agile frameworks which came later, especially those scaling Scrum specifically, and therefore there would have been improvements to improve its weaknesses. Looking at LeSS and Nexus, it has been stated that *"both options provide a much better alternative to Scrum of Scrums"* (Coleman, 2018). For these reasons, as part of the main framework discussion and comparison, it was decided to exclude SoS in the analysis.

Excluding SoS, and the *internally created methods*, which are difficult to evaluate as they are tailored to the environment in which they were created, SAFe, DAD, and LeSS are the most used scaled agile approaches (CollabNet et al., 2018; Kalenda et al., 2018), and therefore were included in the scaled agile framework evaluation and analysis. With the FE enterprise environment being used as the case study for this paper, which is already using Scrum as its non-scaled agile approach, as well as *"the most prevalent agile method used in the transformed organi[s]ations was scrum"* (Dikert et al., 2016, p. 92), Nexus was included in the framework evaluation and analysis as well. The reason for including Nexus, is that, even though it is not a widely used scaled agile approach, it is a specifically *scaled Scrum* approach. With Scrum being a popular agile framework in enterprise, and considering one of the issues regarding scaling agile is resistance to change (as stated in section 2.2), looking into a framework which utilises what the environment already knows and uses, would imply less change and therefore less resistance to the framework implementation, increasing its chances of success. For the evaluation process, the background and general information of each framework is given in paragraph form, followed by Table 6 which lists the framework's pros and cons.

SAFe is a scaled agile framework which combines Scrum, Lean, Kanban and a mix of agile methodologies (Agile, 2018; Ebert et al., 2017), shown to the public in 2011 (Uludağ et al., 2021). There are various levels of SAFe; Essential SAFe, Large Solution SAFe, Portfolio SAFe, and Full SAFe, which cater for different enterprise needs with different complexity levels, providing an enterprise scale solution, specifying the multiple levels with clearly defined roles, processes and organisational structures (Agile, 2018; Kalenda, 2017; Paasivaara, 2017). At a program level, SAFe uses an Agile Release Train (ART) which adds another level to a sprint cycle in order to ensure the team is working in unison. At a portfolio level, it goes one level further, to optimise value streams in order to assist in epic prioritisation which then gets encapsulated within the Program level (Agile, 2018; Francino, n.d).

LeSS is a scaled Scrum framework with two variations, LeSS and LeSS Huge, to cater for a different number of agile teams involved on one project. LeSS can be used when there are two to eight teams and LeSS Huge can be used at an enterprise level for a number of teams which

is greater than eight (Company, 2019; Kalenda, 2017). LeSS utilises Scrum but does not just modify it for a larger number of teams, it aims for the enterprise to apply the correct Scrum components, such as principles and rules, in the simplest method, addressing various agility issues (Coleman, 2018; Company, 2019). The LeSS framework was first published in 2007 (Uludağ et al., 2021).

DAD is a mixture of Scrum and Lean methodologies which focuses more on the delivery of the IT solutions, taking into account the technical aspects, it is goal-driven and is oriented around people and their learning (Agile, n.d.), published in 2012 (Uludağ et al., 2021). Due to it having a larger scope than Scrum, it also contains more roles, which fall into two categories, primary and secondary (Agile, n.d.). DAD is quite flexible as it provides guidelines for six different delivery lifestyles, including agile, Lean, continuous delivery (agile), continuous delivery (lean), exploratory/lean start-up, and program. Each lifestyle is associated with various characteristics such as agile style (Scrum, Kanban, Lean, etc.) (Agile, n.d.; Francino, n.d).

Nexus is Scrum, but executed in a scaled manner. Utilising Scrum's elements and fundamentals, Nexus can be used when there is a project (with one product backlog) with three to nine Scrum teams working on the project (Coleman, 2018; Prikladnicki et al., 2016; Schwaber, 2018; Verma, 2017). This framework came to the public in 2015 (Uludağ et al., 2021). Nexus focuses on the collaboration and dependencies between the Scrum teams, with the main aim of delivering an *integrated increment* every cycle (known as Sprints); and also focuses on quality of deliverable and the speed at which it can be delivered (Coleman, 2018; Schwaber, 2018; Scrum.org, 2017). To mitigate inter-team dependencies and promote transparency, it uses a Nexus Integration Team to communicate and work with the Scrum teams in order to ensure that the Sprint goals and needs for the integrated increment are understood (Schwaber, 2018).

The literature review indicated a lack of assessment criteria when attempting to select which scaled agile framework is best for an enterprise environment (as stated in section 1.8), *"Many noted the lack of any assessment model for conducting such a comparison to guide critical decisions on adopting specific large-scale agile frameworks"* (Conboy et al., 2019, p. 3). That is the reasoning as to why the frameworks chosen for evaluation were compared according to their pros and cons, complexity, and level of inter-team communication, as indicated in Table 6 and discussed further in section 2.4.

The comparison results are based on sources that were extracted for the SLR, triangulating the claims by using more than one source per claim, increasing its validity of the comparison, and thereby providing data that can be used to make an informed decision when selecting an appropriate scaled agile framework. Success factors and challenges that should guide the decision-maker are also included.

Aspects that contributed positively to enterprises successfully scaling the agile environments were coaches and training (both internal and external), standardization of tools and processes used across all teams along with support amongst the teams, executive sponsorship, and being cautious when implementing all the changes in order to scale (CollabNet et al., 2018; Kalenda, 2017). Elements that negatively affected scaled agile adoption were a clash of the traditional organisation culture with the agile values, resistance to change, and lack of support in terms of management and training (CollabNet et al., 2018; Kalenda, 2017).

Table 6: Scaled Agile Framework Evaluation and Analysis

	SAFe	LeSS	DAD	Nexus
Pros	• More defined roles (pro for management) (Larman et al., 2010). • Quicker time-to-market, shortened lead times, and regular product deliveries (Agile, 2018; Uludağ et al., 2021). • If scaled agile is needed and the enterprise still uses traditional methods, it provides an easier transition into agile due to its structure (Francino, n.d). • Flexible in order to support small to extremely large enterprises (Kalenda, 2017; QASymphony, 2018).	• Lightweight, flexible, and easy to adapt to enterprise environment (Ebert et al., 2017; Kalenda, 2017). • Extends Scrum to scale level but not overly complicated (Kalenda, 2017). • Customer-centric, which leads to happier customers and thus better business (Company, 2019) . • Ideal for single product development environment (van Wessel et al., 2021). • Offers customer and business value (Uludağ et al., 2020).	• Flexible and adaptable (Agile, n.d.) • Shorter delivery times (Agile, n.d.). • Optimisation for the entire organisation and not just a certain team/group (Agile, n.d.). • Gives good guidance in terms of which processes would suit a project best, architecture, and development operations (Dolman et al., 2017; Francino, n.d). • Allowing for increased enterprise vision and perception (Uludağ et al., 2021).	• Increased visibility and decrease in inter-team dependencies (Scrum.org, 2017). • Decreased lead time (Scrum.org, 2017, 2018a, 2018b). • Lightweight and easy to setup (Coleman, 2018; Scrum.org, 2018a). • Quicker feedback cycles and continuous integration (Uludağ et al., 2021). • Enhancing accountability with more transparency (Uludağ et al., 2021).

	SAFe	LeSS	DAD	Nexus
	• Keep business goals in focus and increasing employee happiness (QASymphony, 2018; Uludağ et al., 2021). • More widely applicable and caters for highly complex organisations (van Wessel et al., 2021). • Better quality software (Uludağ et al., 2021).			
Cons	• It is one of the most complex of the scaled agile frameworks (Ebert et al., 2017; Uludağ et al., 2021). • It is very structured, which leads to an increase in hierarchy and a decrease in flexibility and agility, making it less adaptable.	• Due to the minimalistic approach, there is less structure within the organisation and more reliance on the mindset and communication between people (Company, 2019; Uludağ et al., 2021). This can be	• Not very descriptive in how to do certain elements, this can lead to disorganisation (Dolman et al., 2017).	• It is still new, so it is still adapting and changing, this can be seen as a negative due to issues not having been picked up yet (Dolman et al., 2017). • Lack of knowledge/understanding

	SAFe	LeSS	DAD	Nexus
	Some have referred to it as the “new waterfall” (Denning, 2015; Dolman et al., 2017; Ebert et al., 2017; Francino, n.d; QASymphony, 2018). • It is very high level as it keeps the enterprise vision in mind, but this can cause longer planning and therefore cycles (QASymphony, 2018). • There is a lot of guidance as to how to implement and start SAFe, but very little guidance once it is implemented (Conboy et al., 2019; Dolman et al., 2017). • Requires strong leadership to be used successfully (van	seen as a high-risk, and negative factor. • It is aimed for more advance practitioners (Coleman, 2018). • Does not work well in complex environments (van Wessel et al., 2021).		of the framework (Uludağ et al., 2021)

	SAFe	LeSS	DAD	Nexus
	Wessel et al., 2021). This can be an issue if there is lack of management buy-in.			
Complexity	High (Ebert et al., 2017)	Medium (Ebert et al., 2017)	Medium (Ebert et al., 2017)	
Inter-Team Communication	High (Dolman et al., 2017)	High (Dolman et al., 2017)	High (Dolman et al., 2017)	Medium (Dolman et al., 2017)

2.4 Discussion

The SLR answered three secondary research questions, namely Questions 3, 4 and 5. This section synthesizes the answers to the three secondary research questions.

The SLR answered Question 3, i.e. *Does the problem instance feature as a class-of-problems in existing literature?* Section 2.2 identified a class-of-problems, namely *the need for scaled agile solutions*, which can be used in large enterprises working on big projects. Included in this problem are the issues which surround scaling agile. In short, the key issues on scaling agile are: (1) management of a large number of people/teams, (2) lack of visibility and control of the project for management, (3) absence of requirements analysis and architectural planning, (4) coordination between teams, (4) maintaining the same standard of coding among all teams, (5) need for documentation, (6) resistance to change, and (7) understanding of the *bigger picture* from each independent team. These findings link very closely to what was discovered in Fourie & De Vries (2017). Some of these are in line with what has been discovered with the FE when embarking on the large banking project, especially with regards to the coordination between teams, which is what has been stated as a key problem at the FE. Scrum.org (2017) states, *"while the teams were working on one product and had one Product Owner, they suffered from being organized as component teams and dealing with cross-team dependencies,"* which confirms the class of problem within the FE and their need to scale agile.

In section 2.3 the suggested areas of solutions were discussed and there were four scaled agile frameworks (SAFe, LeSS, DAD, and Nexus) evaluated in terms of their pros, cons, complexity, and level of inter-team communication; answering secondary research question, Question 4, i.e. *What knowledge areas could be useful to develop a solution to the class-of-problems?* Due to the lack of assessment criteria for scaled agile framework selection, this information was to be paired with the challenges and success factors of scaling agile in order to make an informed framework selection.

Challenges and success factors are learnt from industry implementation and learnings; therefore, the secondary research, Question 5, i.e. *What was learnt, in literature, from the implementation of scaled agile frameworks?*, was also covered; and the findings are synthesised in Table 6. Since the FE selected Nexus as an appropriate scaled agile framework for experimentation, our discussion refers to Table 6, differentiating Nexus from other scaled agile frameworks. We introduce each framework, highlighting prominent *pros and cons*, followed by a discussion on how each framework addresses complexity, concluding on whether the frameworks address *inter-team dependencies*.

SAFe is the most popular and comprehensive scaled agile framework, combining Scrum, Lean, Kanban and a mix of agile frameworks. It has various levels and caters for numerous needs; but due to its comprehensive nature has been labelled as a very heavy and complex framework with structure that resembles the Waterfall methodology.

LeSS is Scrum-based and is a lighter framework as well as being flexible due to its minimal approach, but this approach also leads to its disadvantage of having so little structure that it relies on abstract concepts such as mindset and communication (an already identified problem in the FE), which is a high-risk factor.

DAD combines Scrum and Lean methodologies into a flexible framework which goes beyond the Scrum scope to include delivery, thereby ensuring a full-cycle framework. However, DAD does not provide much information regarding a few of its features which can lead to confusion and lack of direction when trying to implement the framework.

Nexus is an extension of Scrum and is a lightweight framework which is easy to set up and it assists in inter-team dependencies, but it is new, and there is a general lack of information on the subject. It could mean that not all of its implementation issues have been addressed, making it slightly risky.

In terms of *complexity*, SAFe is rated as high with LeSS and DAD rating as medium. Nexus cannot be accounted for as there was no information. An important element with respect to the inter-team dependencies, is that of inter-team communication, which all the frameworks score as high, except for Nexus which is rated a medium. However, one of the challenges raised with regards to scaled Scrum solutions is the resistance to change, and the FE is already running Scrum in their environment, which would lead to narrowing the choices down to LeSS and Nexus. The reasoning is that these frameworks are scaled Scrum solutions (scaled agile solutions with regards to scaled Scrum specifically), whereas the others combine additional methodologies, such as Lean and Kanban, and this would imply that no matter which framework (between LeSS and Nexus) was selected, there would be a decrease in the amount of change and therefore less resistance, which would lead to a higher probability of success.

Both frameworks, LeSS and Nexus, are lightweight and flexible, as well as being an extension of Scrum and can be used by about the same number of teams. LeSS has the advantage that it has a high level of *inter-team communication* whereas Nexus has a medium level. Where LeSS is more customer-centric, Nexus is more inter-team collaboration orientated, which aligns with problem area that should be addressed at the FE.

Comparing their *cons*, LeSS has a key disadvantage of being a minimalist approach, leaving the responsibility on the team member to communicate because the correct mindset is assumed

to exist. This can be proven to not be happening at the FE, as communication and collaboration are already identified issues. Nexus may not have all its flaws brought to light due to being in its infancy, but it is said to improve visibility within a project as well as decrease inter-team dependencies (Scrum.org, 2017), which aligns with the main problem area at the FE. Of the four scaled agile frameworks, it would appear that Nexus is best suited as a suggested solution for the enterprise problem instance.

Agile is about a mindset, of both those using it and the enterprise culture. To be effective, it will change the company culture when implemented and thus become part of the culture. Therefore, the agile solutions have to be moulded to suit the environment in which they are implemented, a compromise between the framework guidelines and the organisational culture (Ebert et al., 2017). It is important to remember this fact and take note that no matter which framework is selected, as the best solution for the enterprise, there is no ideal *standard* framework for any environment. In a real-world scenario, such as within the FE being used for this study, the solution selected needs to be adapted and tailored to fit into the enterprise environment and cater for the organisational culture (Ebert et al., 2017; Kalenda, 2017; Paasivaara et al., 2013).

3. Research methodology

This section provides insight and planning with respect to the research methodology selected for this dissertation. The first two sections, sections 3.1 and 3.2, include literature on different research methodologies, focusing on the research methodology chosen for the dissertation. Section 3.3 investigates threats to the validity of data collection, looking at both the literature and this specific study. Section 3.4 investigates rival theories, also exploring the literature and the case study in question. The following section, section 3.5, addresses the data gathering strategies and the persons who were involved in these processes. Section 3.6 is the last section in this chapter, addressing any ethical considerations when conducting the study.

3.1 Literature on research methodology

Case study research will be used as the research methodology for this study. Case study research can be defined as a comprehensive investigation of a case or cases in a real-world context, where behaviour or causal-factors cannot be controlled. Case studies are used to study *"... individual life cycles, small group behaviour, organi[s]ational and managerial processes, ..."* (Yin, 2018, p. 5). According to Yin (2018), there are three conditions which determine what research methodology to use; 1) how the research question is formed (how, why, what, etc.), 2) the need to manage behavioural events and, 3) concentration on current events. Case study research favours a study where the research question is either a 'how' or 'why', the set of events being investigated have recently passed or are still current, where the researcher has minimal control over events; and where direct observation and interviews or persons involved in the actual case are possible (Yin, 2018). The research *"is not method-driven, but question-driven"* (Ylikoski et al., 2019, p. 1).

There are two case study designs: single-case studies and multiple-case studies (also known as comparative case study research) (Hoogervorst, 2018; Yin, 2018; Ylikoski et al., 2019), subdivided into four sections, as seen in Figure 6, where both types of case study designs could have either a holistic or embedded design (Yin, 2018).

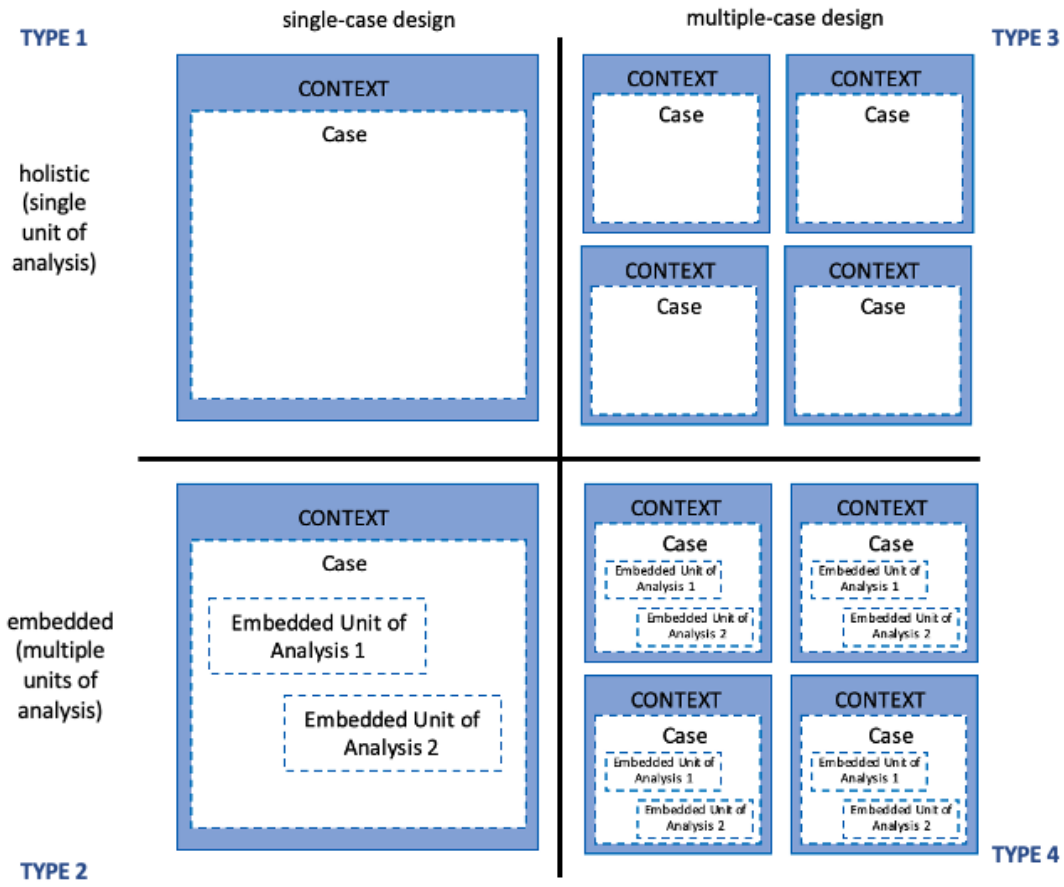


Figure 6: Matrix of different case study designs (Yin, 2018)

For both single-case designs, five rationales of a case are considered: (1) the case could be critical, (2) unusual, (3) common, (4) revelatory, or (5) longitudinal (Yin, 2018). A critical case is one where the case being investigated is crucial to the hypothesis being explored, having very precise conditions. An unusual case is where the study being investigated is something that has not been seen before, or very rarely experienced, such as an unusual medical condition (Yin, 2018). A common case is the opposite of an unusual case, where the case study is about a scenario that is very typical and provides an explanation of an everyday setting. Yin (2018) further explains that the revelatory rationale is used when no previous case was accessible for research. Lastly, when a researcher studies a single case at a different moment in time, this is considered a longitudinal case (Yin, 2018).

The difference between a holistic single-case study and an embedded single-case study is that although they are both about a single case, an embedded single-case study will look at multiple embedded cases (sub-units) within the one overarching system/organisation, providing more complexity (Yin, 2018).

Multiple case studies encompass multiple single case studies, sometimes even being considered a completely separate research methodology to single case studies (Yin, 2018). For multiple case studies, there needs to be some form of replication amongst the cases, either that of the forecasted results will be alike (literal replication) or that they will diverge from each other for predicted reasons (theoretical replication) (Yin, 2018). Due to their comparative nature, multiple case studies are sometimes deemed as more complete, but for a lot of the case study scenarios in single case studies, such as critical and unusual, there would most likely only be one case to research. Multiple case studies are also very resource- and time-intensive, which for a single researcher, is not viable (Yin, 2018).

A case study usually incorporates three main phases: (1) data collection; (2) data analysis and (3) knowledge sharing. Case study research explores cases and information that they provide, using various methods to collect data and analyse it, with the opportunity of drawing findings from the investigation (Ylikoski et al., 2019). The apprehensions pertaining to case study research consider the comprehensiveness of the study, confusion with *non-research* case studies, invalid generalisation, impossible effort, and the possibility of a less distinct comparative advantage compared to that of other research methods (Yin, 2018). However, many research methods, which possibly fall within the case study research methodology classification, should not be categorised as such; due to an incorrect conceptualization of what case study research encompasses (Ylikoski et al., 2019).

3.2 Research methodology for this study

The proposed research methodology for the study is case study research, using a single-case, because the research question starts with *how*, there is little, if any, control over behavioural events, and the research will be focused on a contemporary event, as the digital banking platform went live in recent years and is still in production. Also, the study is classified as a *single case* study as the digital banking platform build, within one enterprise is being studied. Single-case studies can be disputed, lacking the generalisation possibilities, due to only one case being studied. Yet, the purpose of this study is to increase knowledge about the general practicality of Nexus in the Fintech work environment, revealing insights about this single case within a specific context.

The case study incorporates the classic phases of a single-case study, as well as introducing two additional phases to incorporate sufficient context about the main artefact (Nexus) and a demonstration of some of the concepts (at the FE) that form part of Nexus. The phases include: (1) A literature review on Nexus (presented in Chapter 4); (2) A demonstration of specific

Nexus concepts (discussed in Chapter 5); (3) Data collection and analysis (presented in Chapter 6 and Chapter 7); and (4) knowledge sharing (concluded in Chapter 8).

The purpose of the study is to provide an in-depth analysis of the Nexus framework, with the ability to give insights and knowledge into the practical application of Nexus within an existing Fintech work environment specifically focussing on increasing knowledge of inter-team dependencies, aligning with the objective of a case study research which *"is to produce a comprehensive in-depth account of the case"* (Ylikoski et al., 2019, p. 1).

3.3 Threats to validity of data collection

Validity can be described as the level to which something measures what is expected to be measuring (Carmines et al., 2011; Cruz et al., 2019), i.e. the extent that an investigation effectively and correctly answers the question that was posed (Gravetter et al., 2012). There are various types of validity and the use of more than one is required in order to produce a strong case (Cruz et al., 2019), with case study research relying on *triangulation*, i.e. using various means of data to correlate facts (Yin, 2018). Cruz & Berrol (2019) state that while both *reliability* and *validity* are needed for research, validity is of more importance than reliability due to the fact that experiments can produce reliable results but this does not prove validity if there is no agreed definition of what needs to be measured, i.e. validity (Cruz et al., 2019). Reliability is further defined and explored in section 3.3.1.4.

Regarding the various forms of validity, Cruz & Berrol (2019) and Carmines & Zeller (2011) discuss content validity, criterion validity and construct validity, whereas Yin (2018) states that there are four tests to determine the quality of the research done, namely construct validity, internal validity, external validity, and reliability. Literature regards the research of Yin very highly, for that reason following is further explored.

When questions are raised about the validity of the study, the factors that cause these uncertainties can be considered *threats to validity* (Gravetter et al., 2012). According to Yin (2018), tactics can be used to assist case study research in mitigating threats to validity, as seen in Table 7.

Table 7: Tactics to mitigate threats to validity in case study research, adapted from Yin (2018)

Tests	Case Study Tactic	Phase of Case Study Research in Which Tactic is Addressed
Construct validity	• use multiple sources of evidence • have key informants review draft case study report	data collections composition

Tests	Case Study Tactic	Phase of Case Study Research in Which Tactic is Addressed
Internal validity	• do pattern matching • do explanation building • address rival explanation • use logic model	data analysis data analysis data analysis data analysis
External validity	• use theory in single-case studies • use replication logic in multiple-case studies	research design research design
Reliability	• use case study protocol • develop case study database • maintain a chain of evidence	data collections data collections data collections

3.3.1 Literature on the threats to validity

The following sections, 3.3.1.1 to 3.3.1.4, will provide the literature findings regarding threats to validity.

3.3.1.1 Construct validity

To understand construct validity, one must first understand what is meant by construct. The clearest definition of a construct which the researcher could find is the following: *"A construct refers to a concept or characteristic that can't be directly observed, but can be measured by observing other indicators that are associated with it."* (Middleton, 2022) Thus, *Construct validity* is about *"identifying correct operational measures for the concepts being studied"* (Yin, 2018, p. 42), it is ensuring that the tools being used to validate the construct of the study are the correct tools. According to Carmines & Zeller (2011) three practices contribute towards construct validity: (1) the hypothesised relationships between the concepts being measured, need to be stated, (2) observation of the actual relationship between concepts must be explored, and (3) the data from the second step must be analysed in terms of how it is linked to and illustrates the hypothesised relationships between the concepts and thus validates the construct.

3.3.1.2 Internal validity

The next test is *internal validity* which takes into account the valid causal relationships, separating them from relationships where scenarios are not created as a direct result of another condition (Yin, 2018). The aim is to provide evidence that one variable has changed as a result of another variable, while being able to justify that no other variable could have had an influence in the change (Gravetter et al., 2012). For internal validity, a clear connection and tangible data needs to be shown between the variable being studied, and the one that is measured (Cruz et

al., 2019). There is a threat to internal validity if the causal relationship between the two variables is questioned and other variables can be used to provide a different reason as to the changed variable (Gravetter et al., 2012). Extraneous variables are any variables which are not being studied/focused on, but are part of the study. When an extraneous variable changes with the variables being monitored, it provides an alternative reason for the relationship between the two measured variables and this is then considered a confounding variable (Gravetter et al., 2012).

3.3.1.3 External validity

External validity indicates if the findings from the case study can be used in a general context (Yin, 2018), referring to the extent to which similar results would be found if the study was done in another environment/case (Gravetter et al., 2012). Any variable/element that restricts the ability for the results of the study to be produced in another study, can be considered a threat to external validity. According to Gravetter & Forzano, there are, at a minimum, three types of generalisation which could cause a threat to external validity: (1) generalisaition from a sample to the general population, (2) generalisation from one reasearch study to another, and (3) generalisation from a research study to a real world situation (Gravetter et al., 2012). The first generalisation speaks to how well the results would be replicated if this was done with a different group of individuals, with different backgrounds, characteristics, and other aspects similar to these. The second type relates to how well the results can be generalised if the study was done using other techniques, whereas the third generalisation has to do with the degree to which the results can be measured, when for instance considering time passed with accumulated experiences since the study was concluded. Gravetter & Forzano's (2012) summarised version of the threats to external validity is presented in Table 8.

Table 8: Threats to external validity (Gravetter et al., 2012)

General Threats to the External Validity of a Research Study	
Source of the Threat	Description of the Threat
Participants	Characteristics that are unique to the specific group of participants in a study may limit ability to generalize the results of the study to individuals with different characteristics. For example, results obtained from college students may not generalize to non-college adults.
Features of the Study	Characteristics that are unique to the specific procedures used in a study may limit ability to generalize the results to situations in which other procedures are used. For example, the results obtained from participants who are aware that they are being observed and measured may not generalize to situations in which the participants are not aware that

General Threats to the External Validity of a Research Study	
Source of the Threat	Description of the Threat
	measurement is occurring. Also, results obtained with one experimenter might not generalize to a different experimenter.
Measurements	Characteristics that are unique to the specific measurement procedures may limit the ability to generalize the results to situations in which a different measurement procedure is used. For example, the results obtained from measurements taken immediately after treatment may not generalize to a situation in which measurements are taken 3 months after treatment.

3.3.1.4 Reliability

The last of the four tests, *reliability*, refers to replication, i.e. should this case study be replicated by others, following the same process, the results would not differ (Yin, 2018). The assumption made with regards to reliability is that the element being assessed is constant (Gravetter et al., 2012). When a variable, linked to the assessed element is measured, the variable is considered the observed or true score, the measured score is the observed score with the addition of measurement error. The error can be caused by an array of contributing factors and is considered random. Reliability is the measure of this error, with a smaller error factor indicating higher reliability and a lower error factor indicating decreased reliability (Cruz et al., 2019; Gravetter et al., 2012). As already stated, error is considered random and can be caused by numerous variables, some of which include observer error, which is a human error made by the person measuring, environmental changes that affect error if testing is repeated and the environmental conditions differ from test to test, and the participant changes when individuals differ in terms of focus and mood (Gravetter et al., 2012).

Various ways exist to measure reliability, some of which will now be introduced briefly. *Test-retest reliability*, which is the comparison of scores acquired in two tests repeated after each other with the same instruments. If the tests differ, then the test is classified as *parallel-form reliability* (Carmines et al., 2011; Cruz et al., 2019; Gravetter et al., 2012). *Split-half reliability* is when the researchers split the testing tool in half and then compare the scores per individual between the two halves. The closer the scores, the better the consistency and reliability (Carmines et al., 2011; Cruz et al., 2019; Gravetter et al., 2012). The *rater agreement* and *inter-rater reliability* can are seen in literature as transposable, but they should only be used based of the calculations done (Cruz et al., 2019). Inter-rater reliability should be used when there is a calculation done which calculates the actual measurement error whereas inter-rater agreement is based more on the percentage agreement, such as Cohen's Kappa (Cruz et al., 2019). The

inter-rater reliability is the extent to which two observers agree on results after measuring simultaneously (Gravetter et al., 2012). A way to ensure the study is deemed reliable is to document the process undertaken within the study, ensuring a repeatable or reliable study. Table 7, illustrates that a case study protocol and case study databases can be used to ensure reliability when conducting case study research (Yin, 2018).

3.3.2 Threats to validity of data collection within the case study

The purpose of section 3.3.2 is to apply the guidelines that were extracted from literature and discussed in section 3.3.1 and apply them to the case study.

3.3.2.1 Construct validity

During the third phase, *data collection*, interviews and surveys were used as the primary data-gathering instruments, and guidelines from Yin (2018) were followed and referred to Table 8 in order to address threats to validity. To mitigate threats to *construct validity* in case study research Yin (2018) suggests using various sources of data and having main informants review the research report (Yin, 2018). The interviews regarding the Nexus setup, implementation, and running, were done just after the launch of the digital banking platform covering the time pre-Nexus, as well as post-Nexus, to see the changes that had occurred from the Nexus setup and while it had run for a period of time in the enterprise. Interviewees spanned the various parts of the project from Business to Tech, ensuring all perspectives were covered.

One of the challenges of *construct validity* is that suitable concepts should be used. For this study, the researcher had to ensure that a consistent set of Nexus-related concepts were used throughout the study. Chapter 4 includes a review on existing Nexus-related literature. The practical data which has been gathered is to be compared to a theoretical, textbook, framework solution. The Nexus-related concepts being compared are that of the theoretical framework but within a practical environment.

KPAs related to knowledge of inter-team dependencies, are *collaboration* and *communication* amongst the teams. The researcher explored how practices at the FE foster collaboration and communication, using interviews. The results found, after analysing the interview-data, was linked back to the Nexus-related concepts, connecting the practical application to the theory.

3.3.2.2 Internal validity

Through the RCA in section 1.9.2, there has been a positive feedback loop established between knowledge of inter-team dependencies, communication amongst teams, and collaboration amongst teams. An increase in the knowledge of inter-team dependencies can be gauged by the decrease in prolonged inter-team dependent blockers within the project. Effective inter-team

working would be directly linked with increased collaboration amongst the teams, as teams are dependent on each other for various work items. The inability to access these items will block team members from continuing with their work. The only way that there would be better collaboration and knowledge between the teams is through an increase in effective communication among them, implying a valid causal relationship which can be tested when a decrease of prolonged inter-team dependent blockers is measured. Although these blockers were not recorded analytically during the project, this knowledge will be drawn from the interviews of those who worked on the project, which could be seen as a threat to validity. The threat to validity will be mitigated by extracting data from multiple individuals and comparing the opinions extracted.

The adaptation of the Nexus framework was based on opinions about Nexus practises and adaptations that were practical in achieving increased communication and collaboration, linking feedback to the theoretical Nexus-related concepts.

3.3.2.3 External validity

This study was conducted at one company based on a large-scale, but nonetheless single project, as well as the knowledge that a Scrum framework's success is closely linked with enterprise specific factors such as internal change agents and strong drive from leadership (Ebert et al., 2017). The context makes it difficult to determine with verified certainty that the study findings can be used within another domain. Yet, the purpose of the study primarily focuses on identifying the difference between the theoretical components of Nexus and the changes that were required to use Nexus practically within the enterprise, focusing on those practices that enable communication and collaboration. It is believed that these findings would be useful to those within the tech-dominant domain implementing Nexus on a multi-team single product, as the findings can be applied and adjusted to suit the next enterprise considering a Nexus implementation, using the findings, as well as the reasons for the modification, and correlating these back to the specific enterprise to determine what could be considered useful.

3.3.2.4 Reliability

A very strict and methodical protocol was followed for the systematic literature review, so that results would be similar if the same method was followed by another researcher. The study used two analysts to reach intercoder-agreement when coding studies extracted from the existing body of knowledge, as indicated in section 2.1. Regarding the data collected within the enterprise case study, the main threat to the validity, with respect to reliability, would be the observation component, since there was only one researcher who participated in observation,

which may have created some bias. Mitigating observation bias, the primary researcher also used interviews and surveys to cross-validate observations.

3.4 Rival theories

As the analysis of evidence in case study is one of the least developed with this research style, many start without the knowledge of how to analyse the data once they have collected it (Yin, 2018). Yin suggests four general strategies which can be used to assist with this, the first being *relying on theoretical propositions*, which is designing a case study around an already-preserved idea basing the research and literature review on this case. The second is *working your data from the ground up*, which is going through the data one has, and looking for patterns, searching for relationships between data, possibly even using a codebook system to section your data into themes and topics. A third approach is to *develop a case description*, where after having gathered a sizable amount of data but not having established a theme or found suitable concepts while using the code book method, but using the information collected in an informative framework. Lastly, *examining plausible rival explanations*, is the selected approach that will be further discussed hereunder (Yin, 2018).

Although the code book approach was used in the systematic literature survey (Chapter 2), along with research protocols, it is important to consider rival theories, as this strategy compliments the others that were introduced earlier. Numerous rival theories may exist, but attention should be given to the rival theories that are viewed as the most threatening to the case study, investigating the probable rivals but not necessarily all of the rivals (Yin, 2018). Table 9 shows the types of rivals which will be addressed in the rest of this chapter.

As seen in Table 9, rival theories can be subdivided into *craft rivals* and *real-world rivals*, further explained in sections 3.4.1 and 3.4.2 respectively. Sections 3.4.1 and 3.4.2 provide details about the various rival types as individual rivals, but there are cases where more than one rival theory can be found in a single study. Section 3.4.3 looks at the case study in question and the knowledge of rival theories presented, combining them to form a risk mitigation strategy. Section 3.5 summarises the rival mitigation or data-gathering strategy for the different types of rivals for this study, followed by section 3.6 which states the ethical considerations of the study.

3.4.1 Craft rivals

The craft rival is linked closely to the research and design methodology chosen and eliminating methodological artifacts (Yin, 2000). Craft rivals are broken down into the following subsections: the null hypothesis, threats to validity, and investigator bias.

Table 9: List of Rival Theory Types (Yin, 2018)

Type of Rival	Description or Examples
Craft Rivals:	
1. The Null Hypothesis	The observation is the result of chance circumstances only
2. Threats to Validity	e.g., history, maturation, instability, testing, instrumentation, regression, selection, experimental mortality, and selection-maturation interaction
3. Investigator Bias	e.g., "experimenter effect"; reflexivity in field research
Real-World Rivals:	
4. Direct Rival	An intervention ("suspect 2") other than the target (practice or policy) intervention ("suspect 1") accounts for the results *("the butler did it")*
5. Commingled Rival	Other interventions and the target intervention both (practice or policy) contributed to the results *("it wasn't only* me")
6. Implementation Rival	The implementation process, not the substantive intervention, accounts for the results *("did we do it right?")*
7. Rival Theory	A theory different from the original theory explains the results better *("it's elementary, my dear Watson")*
8. Super Rival	A force larger than but including the intervention accounts for the results *("it's bigger than both of us")*
9. Societal Rival	Social trends, not any particular force or intervention, account for the results *("the times they are a-changin")*

3.4.1.1 The null hypothesis

The null hypothesis is a statistical hypothesis in which the researcher suggests that there is no relationship between the cause and effect of observed variables (Yin, 2000). The null hypothesis provides for a hypothesis which the investigator attempts to discredit. However, the null hypothesis leads to an alternative hypothesis, which is the opposite of the null hypothesis and shows what the researcher considers to be the cause-and-effect variables (Vitral et al., 2013). Two types of errors need to be considered. Type I is the dismissal of the null hypothesis which is possibly true, and Type II is the acceptance of a null hypothesis which is potentially false (Yin, 2000). Figure 7 shows the error types and statistical logic of the null hypothesis in a clear visual image. According to Yin (2000), statistical application will only encompass a small extent of the essential social influences, along with the opinion that the null hypothesis does not assist in understanding realistic rivals but rather in the preliminary understanding of proving if a result exists.

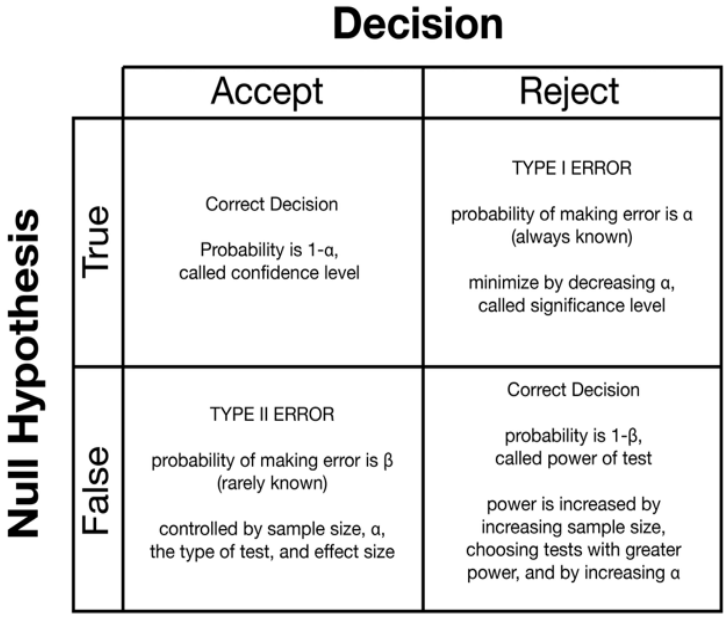


Figure 7: Error Types Linked to the Null Hypothesis (Holland, 2021)

3.4.1.2 Threats to validity

Threats to validity have been termed *rival hypotheses or alternative explanation* (Yin, 2000) and are related to addressing the elements and the manner data is gathered and analysed, already covered in depth in section 3.3.1.

3.4.1.3 Investigator bias

Investigator bias is when the researcher's own personal views or theories subconsciously, but directly, impact the investigation and affect the study, also known as the *experimenter effect*.

In field studies, an added component to this craft rival theory is that the researcher will bring in their bias through the setting, the individuals selected or subtly influencing individuals, known as *reactivity*. Investigator bias is a commonly found rival theory in many scenarios and therefore has to be considered in almost all studies (Yin, 2000). A method to reduce this form of bias is for researchers to state their opinions on the study supported by the theory of what is being researched at the start of the investigation, as it is these opinions which urge the researcher in a certain direction when making decisions in their investigation (Yin, 2000).

3.4.2 Real-life rivals

A real-world rival deals with the cases where the information that does not fit the primary hypothesis but rather competes with it in order to give alternative explanations to the practical results found, leading to the real-life rivals (Yin, 2000). Considering real-life rivals, there are scenarios where an individual cause cannot be credited for the results observed and this increases uncertainty, but the conditions relate more closely to reality. The purpose of the real-life rivals is not to ignore or try to forthrightly disprove them, but rather for the researcher to try to support them. Finding the lack of evidence to support the alternative explanations then leads to strengthening the original hypothesis but also provides a space for the researcher to use the data and the findings to distinguish analyses in between both (Yin, 2000).

3.4.2.1 Direct rival

Direct rivals are those which argue that the observed results are due to an alternative solution rather than the one solution being focused on, regarding both practice or policy solutions (Yin, 2000). According to Yin (2000) an essential element of case study research and collection of data is the collection of information surrounding direct rival practices and conditions. When more rival cases are explored, there is a decreased need for comparison with other cases. With regard to public policy, it is significantly more difficult to provide a verdict as proof (in support or opposition) against the initially targeted solution due to the complex nature of policies that are combined with other policies and not employed in a standardised manner, providing for a lot of variables and make it extremely difficult to trace direct rivals (Yin, 2000).

3.4.2.2 Commingled rival

The commingled rival is where the alternative solution is not disputing the investigated solution, but rather showing that during the same time there was a solution running in parallel with the studied solution, and it could have influenced the results (Yin, 2000). If the investigated solution is part of a program or was implemented alongside similar solutions that aimed for the same results, a full investigation needs to be done to determine whether this alternative could also

explain and support the evaluated change (Yin, 2000). In highly supported community-based projects, various schemes can be implemented at the same time and this makes it challenging for the researcher to credit the changes/results due to an individual factor (Yin, 2000).

3.4.2.3 Implementation rival

The way a solution is implemented can directly impact the results produced, called an *implementation rival* and this may be significant (Yin, 2000). *Implementation failure* is when a variable that affects the implementation was employed unsuccessfully, such as a lack of time or mismanagement that lead to credible reasoning when justifying results which were not expected. Implementation failures are easier to explain than presuming that insight and learnings have been discovered about the solution itself (Yin, 2000). Perfect implementation is an issue in practical life and therefore these failures are vital in a proper evaluation (Yin, 2000). Likewise, there is a concept called *implementation fidelity*, which is proving that the practical (real-life) solution corresponds with the theoretical solution but the results still are found to not match the original prediction, causing this to be an acceptable rival that needs to be focused on (Yin, 2000).

3.4.2.4 Rival theory

Rival theory is when the researcher highlights various perspectives, which differ conceptually, but speak about the same topic (Yin, 2000). These would be perspectives which have possibly been well researched and documented, potentially having influenced the subject matter already, bringing these different viewpoints together, comparing them and analysing them (Yin, 2000). The evaluation does not need to support or disprove one of more theories, but rather could just be an evaluation of the competing theories to produce evidence-backed conclusions that support a single theory of multiple theories, producing research which could contribute to new, or a changed understanding of the topic being investigated.

3.4.2.5 Super rival

This rival looks at the *big picture*, the larger context of the scenario, and this super rival links this larger context to being the main (or part of the main) reason for the observed results. The researcher needs to have an understanding of that larger context (Yin, 2000). The broader context may not be the sole reason for the observations, but it has a big enough influence that it becomes a rival theory to the proposed solution, as the results could have been influenced by both, i.e. not the proposed solutions alone.

3.4.2.6 Societal rival

The societal rival grows on the aforementioned super rival by not just looking at the larger context of the scenario but instead looking at the overall times when the observed changes occurred (social, political, economic, etc.) (Yin, 2000). The societal rival is independent of any focused solution, but rather a result of the changes in the society as a whole where the changes are occurring, thus becoming a rival theory to the focused solution (Yin, 2000).

3.4.3 Risk mitigation strategy

Up to this point, section 3.4 has focused on the literature addressing rival theories. Section 3.4.3 uses the knowledge gained in the previous sections of 3.4 and applies it to this case study, in order to actively try to determine any rival theories in this study and mitigate the risk of them by acknowledging them and planning for them. Threats to validity within this case study have already been discussed in section 3.3.2 and will not be repeated in this section. Table 10 summarises the possible rivals that exist for this study that may contribute towards the study results.

Table 10: Rivals within study and methods to mitigate them

Type of rival	Instance of rival for this study	Rival mitigation or data-gathering strategy
Investigators bias	The researcher was part of the core team which led the Nexus implementation and implementation of it. Investigator bias could be present in probing interview questions.	The researcher is stating their opinion that the KPAs (communication and collaboration) being evaluated improve knowledge on inter-team dependencies and that Nexus has an integral role in increasing the KPAs.
Direct rival	Not Applicable. During the digital banking platform build, when Nexus was implemented, there were no other practices or policies put in place.	
Commingled rival	As stated above, not applicable as no other practices or policies put in place.	
Implementation rival	Implementation failure can occur when the solution is mismanaged. One of the reasons frameworks fail, is that they are not fully or properly supported by higher management to drive the change	This rival will be investigated in the one-on-one interviews where the learnings as well as what worked and did not work are raised. Questions such as “What factors do you think, if changed, would have

Type of rival	Instance of rival for this study	Rival mitigation or data-gathering strategy
	needed among all those using the new solution.	eased some of the issues" would be able to pinpoint if there were failures in the implementation.
Super rival	Nexus was implemented only within the digital banking build, but the company did have other projects being worked on at the same time. The technical team working on the banking build was not working on any other build but this build, therefore keeping the Nexus implementation and results within this product's development.	The researcher was part of the core banking team as well as the Nexus Integration Team, therefore understanding of the *big picture*. The context of the scenario has been considered by the researcher, but the context does not extend outside the digital banking build.
Societal rival	At the time of the digital banking build there were no obvious societal changes occurring.	

3.5 Data-collection strategy

As the Nexus framework is implemented and running, one-on-one interviews were conducted with the key individuals who were directly involved in the digital banking platform build on a leadership or managerial level, and who had been interviewed for the problem gathering phase, as they all have worked closely in the build and implementing Nexus within their respective roles. There was an additional Scrum Master (SM3) who was interviewed for this data gathering, that was not interviewed for the previous round, as they were not in the FE when Nexus launched but they were a key role as they were helping run Nexus within the banking build when they did join the enterprise. These will provide an understanding of the attitudes and opinions of the team members towards the framework implementation, highlighting the relevant pros, cons, and learnings. Technical leads managed the framework as well as the people involved, and have strategic knowledge into what has and has not worked within the Nexus framework.

The reasoning for selecting the same group of people to interview for the pre-Nexus phase and well as the post-Nexus phase is to obtain an end-to-end understanding from those who were part of the implementation as well as leading the facilitation of the framework. The core banking team and SMs, besides being heavily involved in the project build, have all been well-exposed to Scrum and the agile way of working. They have knowledge of how Scrum teams work,

provide feedback about the approach and implementation of Nexus with sound background knowledge.

With so many systems forming part of a single project, it was decided to have an overall PO of the entire project, but due to the size of the build and number of systems involved, a decision was made that the Business Analysts would assist in owning and managing the build of the systems which they designed and were accountable for. This was to ease the management of the build as the Business Analysts were tasked with designing, as well as working constantly with UI/UX and developers, until the system was complete. The list of candidates chosen for the study were:

- Key individuals and management:
    - Product Development Lead (PDL)
        - Product development lead for the banking build, Product Owner for the entire banking build project
    - Lead Technical Manager (LTM)
        - Lead technical manager for the banking build
    - Senior Business Analyst (SBA)
        - Senior Business Analyst, Product Owner for the sign-up process and compliance system
    - Scrum Masters
        - Scrum Master 1 (SM1)
            - Scrum Master for the India and Mauritius teams
        - Scrum Master 2 (SM2)
            - Scrum Master for the South Africa team
        - Scrum Master 3 (SM3)
            - Scrum Master for the South Africa and India teams

The author, not listed above, was also part of the core banking build team, with the role as a Business Analyst. Fulfilling the role of Product Owner for the CRM, IMS and EMS systems, their input is provided through observation.

With this group of participants, inclusive data can be collected and analysed to give information on the Nexus approach to answer the main research question about practices that improve collaboration and communication to improve knowledge of inter-team dependencies.

Interview questions were structured, relating to Nexus-concepts (as defined in Chapter 4) and include the following:

- Why was Nexus chosen? What factors were considered?

- How was this proposed to the company?
- How was the adaption to the new system adopted?
- What has worked? Why?
- What hasn't worked? Why?
- What factors do you think, if changed, would have helped eased some of the issues?
- What were the main lessons/learnings taken from this implementation?
- What are the main differences between the theoretical and practical application of Nexus?
- What changed/was adapted from the original implementation of Nexus to the way it was run by the end of the project?
- If you knew everything you know now, would you use Nexus again?
    - If so, what would you change, from implementation to general operation?
    - If not, what would you consider implementing instead?
- In general, what is your personal opinion of scaled-scrum/agile?
- Lastly, do you think Nexus was a success, with respect to this launch MVP bank build?

3.6 Ethical considerations

The study applied principles of ethical behaviour as prescribed by the Faculty of Engineering, Built Environment and Information Technology (EBIT) of the University of Pretoria. With regards to the people being interviewed, there are ethical considerations to take into account, such as the consent of the participants who are involved in the data-gathering techniques (surveys and interviews). Everyone who participated in any of the data-gathering techniques were informed about the study, the background, and research involved, and were requested to sign a consent form giving permission to the author to use their personal opinions and any information given within the surveys and/or interviews as evidence in the study. The enterprise being evaluated, has given their permission to be used as the case study unit and will review the final version of the dissertation to ensure that only approved information about the company is published to the public.

4. Nexus

In this section Nexus, the scaled Scrum framework selected for the enterprise, is described and its various concepts are communicated according to the theoretical model found in literature. In the subsequent subsections, Scrum is introduced, since Nexus evolved from Scrum. Then, Nexus is introduced, the framework components/practices are described, which includes roles, events and artefacts.

4.1 Scrum

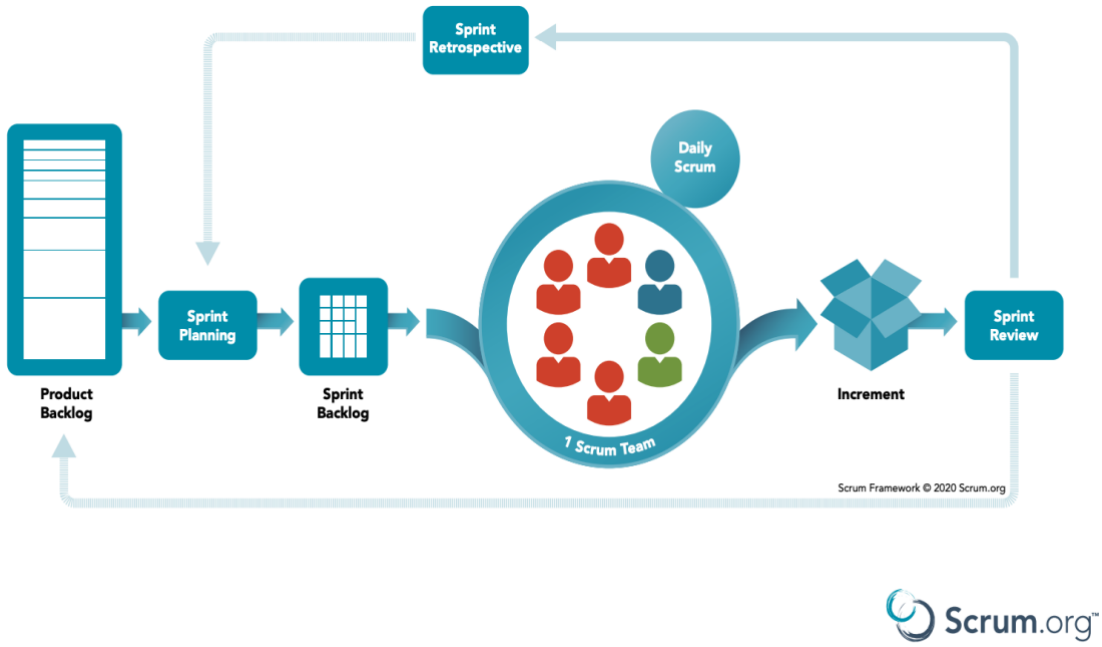


Figure 8: The Scrum Framework (Scrum.org, 2021b)

To understand Nexus, an understanding of Scrum is required, as Nexus evolved from Scrum. Scrum started in the early 1990s based on the ideology that, largely through experience, knowledge is gained (also known as empiricism) (Kurt Bittner, 2018). Primarily focused on product development, Scrum uses iterative and incremental deliverables/goals to learn continuously and therefore improve product usefulness and mitigate risk. It has three pillars which are used to maintain the Scrum process, *transparency*, *inspection*, and *adaption* (Kurt Bittner, 2018). According to Bittner (2018), Scrum is used by 90% of agile teams and its success lies in its simplicity and the concentrated focus of a single team building a single product, with a few roles who all have their own specific focuses within the build. Figure 8 shows the Scrum framework and the various events, this will be further discussed in section 4.3, along with the Scrum roles, events, and artefacts.

4.2 What is Nexus?

Nexus was created for three to nine team projects, using Scrum, and all working on one product (Kurt Bittner, 2018). Nexus should have one PO who uses and builds a single Product Backlog, used by the teams, who have the goal to deliver at least one *Done* increment item for each Sprint (based on the team's definition of *Done*) (Kurt Bittner, 2018; Scrum.org, 2021a). According to the Nexus Guide (Schwaber, 2018), Nexus should solve challenges with scaling Scrum by reducing inter-team dependencies, *making them more evident*, as well as increasing self-management, and guaranteeing accountability (Scrum.org, 2021a).

4.3 The Nexus Framework

Nexus builds off the Scrum framework, using it as a foundation and extending from it (Scrum.org, 2021a). In Scrum, key concepts include: *"three roles (Product Owner, Scrum Master, and Development Team), five events (the Sprint, Sprint Planning, Daily Scrum, the Sprint Review, and the Sprint Retrospective), and three artefacts (the Product Backlog, the Sprint Backlog, and the Product Increment)"* (Kurt Bittner, 2018). Nexus takes these elements and modifies them to create the Nexus framework. These concepts (Nexus along with the Scrum components it uses) are the answer to Questions 6, i.e. *What are the key components which can be compared to evaluate the differences in the practical and theoretical frameworks?*

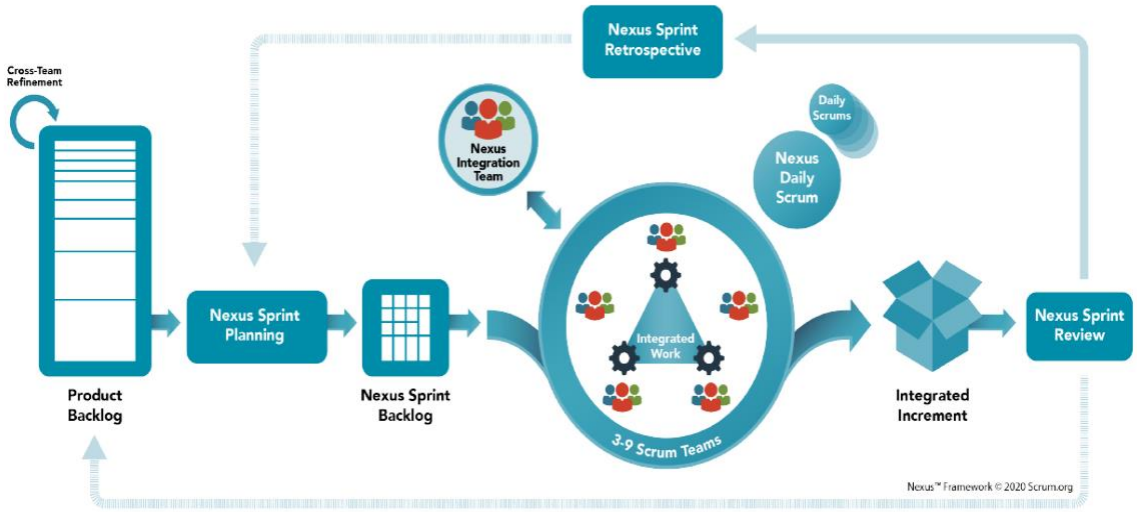


Figure 9: The Nexus Framework (Scrum.org, 2021a)

Comparing Figure 8 with Figure 9 (the Scrum framework versus the Nexus Framework), Nexus expands on Scrum by adding in a new role (the Nexus Integration Team), five new additional events (Nexus Sprint Planning, Nexus Daily Scrum, Nexus Sprint Review, Nexus Sprint

Retrospective, and Refinement), one additional artefact (Nexus Sprint Backlog), and removing one event (individual Scrum Team Sprint Review) (Kurt Bittner, 2018), as indicated in Figure 10, where additional elements are indicated with "*".

Roles	Events	Artifacts
Development Teams	The Sprint	Product Backlog
Product Owner	*Nexus Sprint Planning* *	*Nexus Sprint Backlog* *
Scrum Master	Sprint Planning	Sprint Backlog
Nexus Integration Team *	*Nexus Daily Scrum* *	Integrated Increment
	Daily Scrum	
	Nexus Sprint Review *	
	Nexus Sprint Retrospective *	
	Sprint Retrospective	
	Refinement *	

* Nexus specific

Figure 10: Scrum and Nexus Specific Roles, Events, and Artefacts (Kurt Bittner, 2018)

4.3.1 Nexus-specific roles

The only Nexus-specific role is the Nexus Integration Team (NIT), comprising of the PO, an SM and the various other NIT Members (members from individual Scrum teams as well as the possibility of members from other areas within the enterprise, such as Finance and Operations who join when applicable) (Kurt Bittner, 2018). The purpose of this team is to facilitate in implementing Nexus and to add in a level of visible accountability within the Nexus integration, coordination and functioning, being the central point of the integration and dealing with technical and non-technical inter-team impediments. The NIT has the responsibility that the collective work finalised within the Nexus Sprint has a *Done* integrated increment (Kurt Bittner, 2018; Scrum.org, 2021a).

The PO has the main responsibility of ensuring that the product's value is boosted, and has the responsibility of ordering the Product Backlog, making sure that that it contains the correct items. The SM of the NIT has the role of ensuring that the Nexus framework is understood and acted upon, also acting as an SM of individual Scrum Teams within Nexus. The other NIT members assist the individual Scrum Teams in successfully implementing Nexus, and are ultimately responsible for their team's *Done* Integrated Increment at the end of the Sprint (Kurt Bittner, 2018; Scrum.org, 2021a).

4.3.2 Nexus specific events

As stated in section 4.2, there are five Nexus-specific events: Nexus Sprint Planning, Nexus Daily Scrum, Nexus Sprint Review, Nexus Sprint Retrospective, and Refinement. Practices that

relate to *knowledge about inter-team dependencies* and the two contributing KPAs, namely *inter-team communication* and *inter-team collaboration*, are also highlighted.

Nexus Sprint Planning works like a Sprint Planning event, where the refined Nexus Product Backlog is used to organise and plan the tasks taken in by the Scrum Teams for the upcoming Sprint, planned with the team and PO. The primary objective of this event is to create the *Nexus Sprint Goal*, the objective to be achieved within the Nexus Sprint. The Nexus Sprint Goal is associated with the *Product Goal* and the *Sprint Goals* for individual Scrum Teams (objective for individual Scrum Teams in that Sprint). The creation of a Nexus Sprint Backlog is related to the Nexus Sprint Goal *highlighting inter-team dependencies* (Kurt Bittner, 2018; Scrum.org, 2021a). Its completion status is reached when all the individual Scrum Teams within Nexus have completed their Sprint Planning (Kurt Bittner, 2018).

Nexus Daily Scrum is done with various members from individual Scrum Teams *who come together to discuss* and understand any integration concerns and *inter-team dependencies* with their issues and impacts (Kurt Bittner, 2018; Scrum.org, 2021a). Understanding the status of the integrated Increment and usually discussing what was done by their respective teams the previous day, *new dependencies are identified*, and information which other teams may need to be aware of (Kurt Bittner, 2018). Adjustment of plans, backlog updating, and *inter-team communication* can occur throughout the day and not just within this specific event (Kurt Bittner, 2018; Scrum.org, 2021a).

The next event to be considered is the *Nexus Sprint Review* which takes place at the end of the Nexus Sprint and replaces the individual Scrum Team Sprint Reviews, with all team members of Nexus in attendance. This is because feedback is needed on the complete Integrated Increment from the stakeholders, with the possibility of updating the Product Backlog, as the individual Scrum Teams may not have created an individual Integrated Increment (Kurt Bittner, 2018; Scrum.org, 2021a). This event is done so that in the teams, each teams' stakeholders have the opportunity to comment on what can be done to improve the general Nexus progress. Having one review indicates that stakeholders can attend this review instead of multiple individual reviews. Issues regarding the integrated Product as a whole can be identified, as individual software components may work separately but not when they are combined. Looking at the entire Integrated Increment, the teams remember that they are working on a single product (Kurt Bittner, 2018).

Nexus Sprint Retrospective concludes the Nexus Sprint. During this event, the objective is to reflect on the Sprint that is ending and understand what needs to be changed or retained, to improve the Nexus methodology, making it more efficient and produce higher quality work (Kurt Bittner, 2018; Scrum.org, 2021a). This event consists of the NIT and any other interested

party who can contribute information on inter-team impediments, providing clarity on any issues that were shared between teams. The Nexus Sprint Retrospective allows for issues to raised and investigated with solutions being proposed. Input from the individual Scrum Team Sprint Retrospective is used as the basis of this event, building to more general and overarching issues and feedback (Kurt Bittner, 2018).

Refinement is an event that is not time-boxed, but rather occurs when teams work together to *resolve inter-team dependencies* with the Nexus Sprint. There is continuous refinement of the Nexus Product Backlog based on what items teams can work on with inter-team dependencies being considered. Therefore, various members of the different individual Scrum Teams participate in Refinement. Items are broken down from large, over-arching tasks, down to lower-level sub-tasks to be done within the task, which can be taken in as work items by the individual Scrum Teams. This event's main intentions are understanding what work will be performed by the individual Scrum Teams and which items need to be delivered in what order, as well as *identifying inter-team dependencies* (Kurt Bittner, 2018; Scrum.org, 2021a).

4.3.3 Nexus specific artefacts

An artefact, within Scrum terminology, is a representation of work done and is created to provide a platform to clearly analyse and adjust where necessary (Kurt Bittner, 2018; Scrum.org, 2021a). As stated previously, the *Nexus Sprint Backlog* is the one addition for Nexus to the general Scrum based artefact. Comprising the inter-team dependent items, this backlog is used to understand and emphasise these dependencies, being adjusted daily based on new learnings and providing information on progress made with the Nexus Daily Scrum (Kurt Bittner, 2018; Scrum.org, 2021a). It should be noted that the *Integrated Increment* is the combination of work done by all the Scrum teams (within Nexus) within that sprint to produce an incremental deliverable which is usable and has the potential of being released (Kurt Bittner, 2018).

5. Demonstration

The following chapter explains concepts from the theoretical Nexus and how they were applied and adapted in the project instance, providing an overview of the 'modified' Nexus that was applied within the enterprise. Some of the main differences found were the use of BAs as team/system POs, adding the Sprint Review event back in and removing the Nexus Sprint Review, modifying the Nexus Daily Scrum event to only focus on inter-team dependencies, and the removal of the Nexus Sprint Retrospective and Nexus Sprint Planning. All roles, events, and artefacts, whether modified or not, are discussed and explored in further detail in Chapters 6 and 7. Providing context for demonstrating a Nexus implementation at the FE, an overview according to a timeline, for the case study, is provided.

5.1 Demonstration timeline

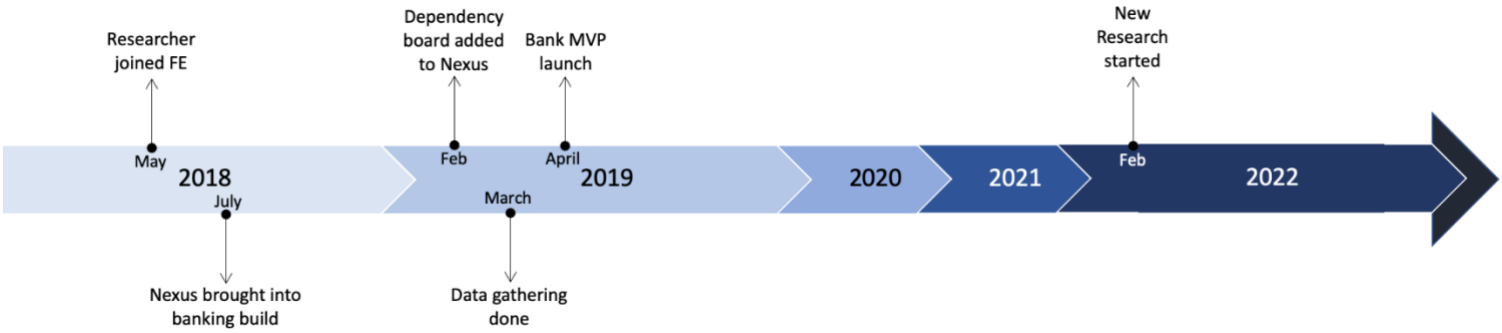


Figure 11: Demonstration timeline

The demonstration timeline, found in Figure 11, shows the timeframe from when the researcher joined the FE, May 2018, to current, highlighting major milestones along the way, such as when the banking build had its MVP launch in April 2019. When then the researcher joined the project, the enterprise had already implemented Scrum, but it was very team-centric and run in an isolated manner, as described in section 1.9. In July 2018, Nexus, a scaled agile solution, was brought into the FE for the digital banking platform build as a solution to this issue, aiming to improving inter-team collaboration and communication. Only in February 2019 was the Nexus Daily Scrum changed to use the dependency board, a major success for the Nexus implementation, and discussed in more detail within the following section. Just before the MVP bank launch in April 2019, the data gathering process was conducted, where both pre- and post-Nexus interview questions were asked to obtain a full spectrum of just how much Nexus had changed from its beginning phases to where it was at the MVP launch, as well as what had and had not worked. The logic behind this was that those being interviewed had been part of the build while the teams had implemented Scrum, and had been part of the transition and/or

running of Nexus, making them ideal candidates to ask about the Nexus implementation as they had insight to how much had changed and what was going on within Nexus. The researcher then had to pause the research in 2020 and 2021 due to various reasons, both environmental as well as personal, but did continue to work for the enterprise. In 2022, the research was resumed, and new data for the literature survey was obtained and utilised. There was a concern that the study may no longer be relevant, therefore the research done in 2022 was also to confirm whether the topic was still relevant and/or if it had already been done by others, but no other research could be found regarding Nexus-specific case studies within a Fintech company. A lot of research has been done on other scaled agile solutions, such as SAFe, but there is still a large gap in empirical data on Nexus and its practical implementation. With this knowledge, the decision was made that the case study is still relevant and would positively contribute to the academic writing within the agile community, as the findings and lessons learnt are still valuable to those wishing to implement Nexus within their environment. The data gathering was done close to the event in question, Nexus for the MVP banking launch, when the events were still new, and this makes the data more reliable than if it was gathered two years later.

5.2 Nexus demonstration

Figure 12 shows the legend for the images used within this chapter, showing the PO as blue, the Lead Technical Manager as red, the SMs as orange, the BAs as green, and the Development Teams as purple with the dark purple icon being the Tech Lead of that development team.

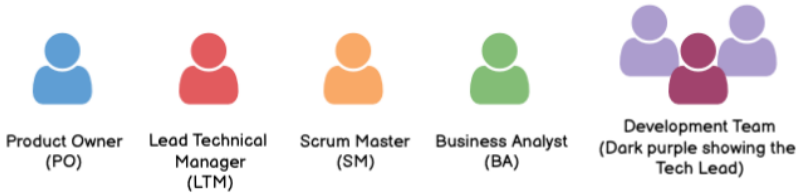


Figure 12: Legend to identify the roles which are represented in the demonstration

Figure 13 shows a visual representation of the banking platform build team, not to scale with the exact number of development teams and members but to scale based on PO, BAs (there were three BAs as shown but one left early on in the banking build, hence they are not part of the data gathering process), SMs, and LTM. The NIT was composed of the PO, the LTM, SMs, BAs, and Tech Leads from each team, shown by the black triangle symbol on the NIT members in Figure 13. There were not enough SMs within the enterprise to be assigned to each team so SMs took on more than one team with no central NIT SM, as indicated in the theoretical framework, but rather all SMs within the enterprise taking on the role of undertaking that Nexus is understood, implemented, and executed.

In the theoretical version of Nexus, the role of a PO, as stated in Chapter 4, is to increase product value as well as to manage the Product Backlog. The PO should be overseeing all the various builds and know, on a high level, what each team is working on, planning ahead. Managing multiple members, as well as a *large build*, is too much for a single individual to do, within the FE. Therefore, the BAs assisted some teams, taking on the role of the PO. The BAs, acting as POs, were the point of contact for the authorised PO regarding updates on a system's build. The

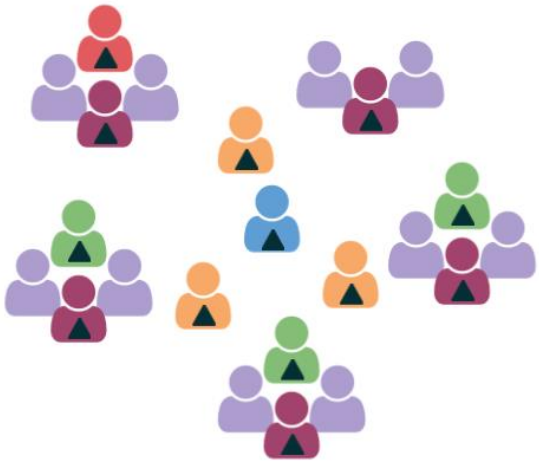


Figure 13: Representation of the banking platform build team structure

BAs also managed the backlog and discussed items that would be brought into the next individual Scrum team's Sprint with the PO. Within the enterprise there was also the LTM who sat within a specific team, mainly dedicated to the web service/APIs used across teams but would liaise with the PO on the backlog items for the team as well as discussions and decisions linked to IT infrastructure/architecture. The LTM was part of the NIT and assisted the PO with more technical decisions and management.

The Nexus Sprint Review was removed from the Nexus implementation at the FE as it was viewed as unsuccessful and the individual Scrum Team Sprint Review was brought back in, which is the opposite of what the theoretical framework states. Instead of one overarching

Nexus Sprint Review, multiple reviews were performed per team, attended by the PO, SMs, BAs, and sometimes other stakeholders. Nexus Sprint Planning and Nexus Sprint Retrospective were removed from the FE's Nexus implementation, as the PO could not prescribe a Nexus Sprint Goal, only Sprint Goals per team, and it was deemed that there was no time to do a Nexus Sprint Retrospective.

A massive success within the FE's Nexus setup was the Nexus Daily Scrum, where after some alterations the enterprise used this event to only highlight inter-team dependencies. The FE used Trello and created a Kanban board showing only items with dependencies and these items were raised, discussed, prioritised, and updated in the Nexus Daily Scrum. The use of a Kanban board, used in this manner, to just show and update dependencies is considered a Nexus Sprint Backlog.

The normal Scrum events, Sprints, Sprint Planning, Daily Scrum, Sprint Review (as mentioned earlier), and Sprint Retrospective, were applied as intended in the individual Scrum teams. These events were applied and managed by the BAs or the SMs of the various teams, any progress, feedback, or issues raised were addressed with the PO. The Product Increment would be aligned with the individual Scrum team Sprint Goal, and would be the incremental deliverable done by that team.

The Sprint Backlogs were managed by the PO in conjunction with the BA, SM, LTM, or Tech Lead of that system/build. A Sprint Backlog was created per team. The items which were coming into the next team Sprint were confirmed in discussions between these members about the Backlog. Refinement of the Product Backlog was not done across teams, as it described in the theoretical framework. Rather, the PO would manage the items in the Product Backlog, sometimes with the assistance of the LTM, to decide on items included for the next team Sprint.

Figure 14 shows a simplified version of what the Nexus structure within the enterprise looked like at the time of the MVP bank launch. Consolidating the previously-noted arguments, Figure 14 indicates that the enterprise implemented multiple full individual team Scrums, which produced multiple Product Increments, where some Increments were integrated, whereas others were stand-alone Increments for the individual system being built.

Using the Nexus Daily Scrum as an event for the NIT to discuss the Nexus Sprint Backlog, the raising and discussing of inter-team dependencies was done, with feedback going back and forth between this event and the Scrum Sprints, as indicated with the bi-directional arrow in Figure 14.

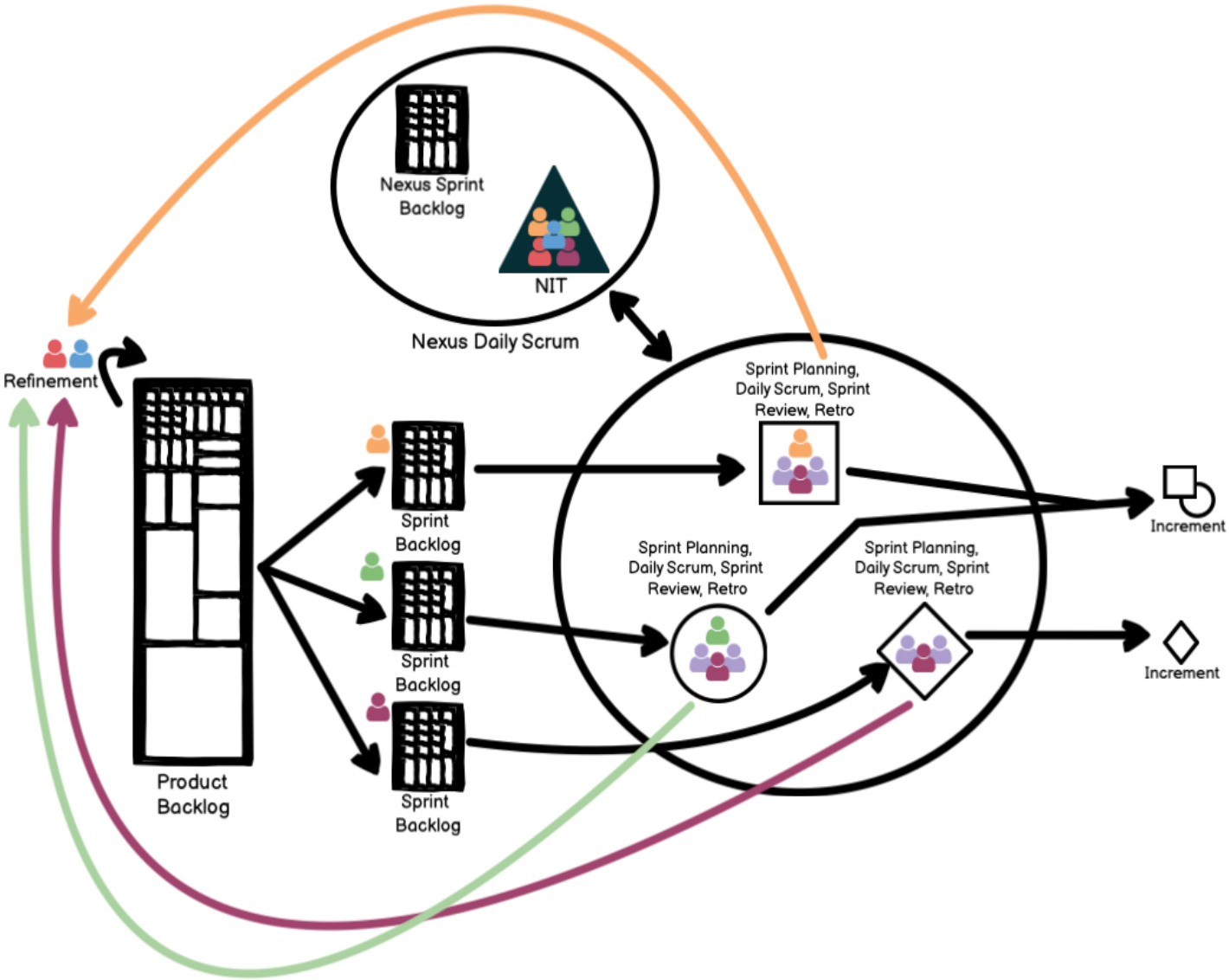


Figure 14: Simplified version of what was implemented within the FE by MVP Bank launch

The coloured arrowed lines in Figure 14 indicate that Individual Sprint Reviews were held to demonstrate the work done in the Sprint and feedback was given to the PO (and sometimes the LTM) to refine the Product Backlog before deciding on individual Scrum Team Sprint Backlogs. These items/features that need to be developed are included in the Scrum Team Sprints through the BAs, SMs, and/or Tech Leads, and the cycle starts again. As stated previously, Figure 14 provides a simplified version of the structure within the FE. The actual implementation included more than three teams and many more members were involved.

6. Results

Through the interviews and observation by the researcher, the following chapter investigates what the deviations were between the practically implemented Nexus within the enterprise and the theoretical framework. This consists of two sections, section 6.1 that states all the components that were implemented and modified within the enterprise, as well as those components that were *not* implemented. Section 6.2 briefly presents items from the Nexus framework that were implemented but not modified in the practical case study. These two sections will answer the secondary research question, Question 7, i.e. *What were the reasons that the theoretical key components did or did not work within this enterprise's environment?* It should be noted that in the quotes, when 'Stand-Up' is mentioned, as used by the FE team members, it is a synonym for Daily Scrum. Sections 6.3 looks at results which provide learnings about the Nexus framework within at the FE, looking at enterprise context which affected how Nexus was utilised. Lastly, section 6.4 concludes this chapter with information that is *not* Nexus-framework-related, but rather related to general lessons learnt within the case study.

6.1 Modifications between theoretical and practical Nexus

All the modifications between theoretical and practical Nexus are stated in the demonstration (Chapter 5) of this dissertation, but this section elaborates on the reasoning and examples collected during data-gathering, to affirm these modifications. The main modifications include: (1) using the BAs to act as team POs, (2) the removal of the Nexus Sprint Review and reinstating of the individual Scrum Team Sprint Reviews, and (3) the Nexus Daily Scrum becoming purely a dependency-highlighting meeting. All other modifications are found in the last subsection of this section.

6.1.1 Using the BAs as POs

With over 50 team members working on various items simultaneously, the overseeing of the various builds can become overwhelming for a single individual, the PO, to manage and control all elements of the project. The workload problem of the PO was highlighted by the LTM, "*[i]t is possible to do it through the Scrum Masters if they are there as well, usually it would be the Product Owner; so, if there are multiple teams working on something then the Product Owner would attend those planning sessions and say what the dependencies are. With this many teams though, it's not always easy for one product owner to do all of that.*" Within the FE, the PO role was partially taken over by some of the BAs, with a few of the builds having BAs work on them. Their role was to gather business requirements, translate those into system design specifications, present the solution to business and then present the approved solution to the

development team, finally leading the development team to build the system (including testing and deployment support). As stated by Scrum Master 3 (SM3), "*[w]hat I do really like, and it is only being done with two teams is, there is a Nexus Product Owner, and then there is a team Product Owner that sits with the Nexus Product Owner, this frees up the Nexus Product Owner to focus on the bigger picture and it does not create favouritism toward that Product Owner to one team, it does not neglect any team. What happened here is that we had BAs that were helping our teams and inadvertently became their Product Owners, working with the Nexus Product Owner, but that is only happening with two of the five teams. I would try to make that happen with all five teams and then have the Nexus Product Owner liaise with those 5 Product Owners. It would almost be like a starfish, with the Nexus Product Owner in the centre and the tentacles being the team Product Owner.*"

Having been part of their specific systems' build from conception through development and to deployment, made the BAs the ideal people to assist the PO with managing the builds by handling the build's backlogs and informing the PO of the progress while discussing the items that were scheduled for the next Sprint. This process eased the pressure on the PO and helped manage the builds while allowing focused management and direction on each team, with a single point of contact for the PO which made communication and understanding of each build better.

6.1.2 Removing the Nexus Sprint Review, adding the individual Scrum Team Sprint Reviews

The individual Scrum Team Sprint Review is an event which is removed from the Nexus framework and is replaced with the Nexus Sprint Review. The theoretical concept of the Nexus Sprint Review is to bring all team members together, including other stakeholders, and therefore all can attend one review rather than many reviews. For the enterprise, putting this into practice was not successful, and the FE resorted to having individual Scrum Team Sprint Reviews at the end of each team's Sprint.

Through observation, the researcher, who attended the one Nexus Sprint Review that the FE had, can state that this specific format/attempt at a Nexus Sprint Review was not successful. It took a few hours and there were too many people present whose incremental deliverables were not directly linked to items that were reviewed, as not all increments were integrated with each other. The practical version at the FE differed from the theoretical version of an Integrated Increment resulting in a loss of concentration and time wasted. Those members who were not present in South Africa, i.e. the Mauritius and India based teams, also had difficulty engaging with those sitting together in South Africa. Due to the size of the project not everyone had a full

understanding and therefore, a vested interest in every aspect being built, therefore reducing the size of the Nexus Sprint Review could have achieved more, as eluded to by the SBA when they stated, "*[m]aybe the integration between the smaller teams to Nexus should be better because it's a massive project and you can't have an overview of everything that is happening. So Nexus is kind-of like the overview and it seeps down into smaller teams, and I don't know how that could be improved. Because I think, the way that we're doing it is fine, the scope of this project is just so crazy that having your fingers on everything all the time is close to impossible*." The FE only attempted the Nexus Sprint Review once with the possibility of its premature removal, as stated by SM1, "*[i]n that whole time-period we only did one Nexus-level Review; we should have been getting better every three weeks, and if we had done that, we would have been a lot better off. That's the core of Scrum; inspect and adapt*." The Nexus Sprint Review could have been applied in a better way and had additional attempts been made to better inform the decision to remove the Nexus Sprint Review, the possibility of there being improvements would have existed.

Each team would end their Sprint with a review and demonstrate what had been achieved within the Sprint, showing this to the BA, PO, and SM, and occasionally to an external stakeholder. The Sprint Review in itself brought in some challenges when stakeholders and end-users would not see a review for an extended period of time and when a review was presented, the partial product would be much bigger. Multiple changes would then be required, moving away from the incremental building approach of fail fast and iterate quickly. One such example was given by SM3, "*[a]n example we experienced today was having the stakeholders be involved with demos more regularly. So not getting to a point where development has been going on for four to six months and the end-user hasn't seen the product, and come basically go-live, the end-users actually see the product and raise a whole field of concerns. This doesn't directly link to scaling-scrum, but it would directly link to scrum, which is the foundation/fundamental part of scaling scrum. One of scrum's fundamentals is to have continuous feedback, have the end user and stakeholders ultimately signing-off and giving feedback, improving the product, and raising potential issues on a very regular basis This is something we have had to a certain extent but not to an end-user extent, we've had it to a business-partner extent, but not to an end-user extent; and that would have helped, I think even when we go-live we will see that*."

6.1.3 Using the Nexus Daily Scrum as a dependency-highlighting event

The most spoken about event and something reiterated in most of the interviews was the Nexus Daily Scrum, as this was one of the most successful events within the FE, maturing within 6 months of initiating the Nexus in the FE. This was the case because, "*[w]hat works is, on a project like ours, the scale at which we work at, you do need that daily collaboration between*

all leads on understanding dependencies," states the PDL, and that is what the Nexus Daily Scrum became within the FE. According to the Nexus framework, as clarified in Chapter 4, the Nexus Daily Scrum is an event where members from various teams meet and discuss integration concerns, inter-team dependencies, the status of what had been done the previous days within their individual team, and information that other teams should be made aware of. When the enterprise initially set up these meetings, they were treated as Scrum Team Daily Scrums where each member informed the other teams what their team had worked on the previous day, work that was planned for that day, and any issues they were facing. This lead to a very long Nexus Daily Scrum meetings and the inter-team dependencies were neglected due to an information overload. The nature of the Nexus Daily Scrum thus changed to only highlight the inter-team dependencies, as highlighted by SM2, "*[w]e adopted the same way we did it in the normal Scrum framework, it wasn't a status update, it was more of a collaborative discussion around the goal; what did I do yesterday, what am I going to do today, what are my blockers? That just didn't work at a Nexus level because that wasn't the bottleneck or the problem, the problem was that there were teams that were interdependent, that needed to build features together and were simply waiting on one another, there was a lot of obscurity and grey-areas as a result. We needed to reduce that, and the Nexus Daily was the perfect opportunity.*" SM2 went on to say, "*[w]e took a while to learn from that, that is why we had to bring the dependency and integrations into the Nexus daily Scrum approach, because saying 'what I did today' and 'what I did yesterday' and my impediments, but rather what was going to make us grow as an entire team, was going to help us unblock individuals. If we had learnt that earlier, we would have been much better off from the beginning*,"

Instead of every NIT member speaking because they felt they had to say something, team members only spoke up in this event if they had a dependency linked to another team which would become, or had become, a blocker. The LTM confirmed this by saying that "*[t]he big change was, when we started Nexus, people were treating it as if it was sort of an abbreviated Scrum-Of-Scums, but then we changed that, we got the Trello board in, and now it's more of a discussion of what is happening. In a Stand-Up, generally everyone would have something to say, now in the Nexus it's just a matter of asking if you are dependent or blocked by anything, so a lot of the teams now just say no we are okay. When we started, every person felt like they were obligated to say something, because there has to be something happening kind of thing.*" SM1 also confirms the guidance provided by the LTM, by saying, "*[i]f you are doing your daily Stand-Up, find a format that makes blockers and dependencies visible fast. State what your blocker is and assign someone to help, quick. Find that first. In a daily Stand-Up it's really not about what you are doing and your progress, it's about "How am I blocked and who's blocking me?" and getting that person in the daily Stand-Up of the other team to commit to*

assisting you and getting it out of the way. A probing question was used, asking the SM1: Are you talking at a team level or Nexus integration level? The SM1 responded as follows: *"Nexus integration level, because what would happen is one team would be blocked and they would stand there and state that they were blocked but didn't do much about it. It took us a very long time, we went running around for eight months and now, in the last two months, we have got this blocker-dependency board and that has helped us, that has given us the visibility. So that's what we should have done that earlier. Make that visible fast, get your stand-up to work in that fashion.*"

The Trello board, shown in Figure 15, shows the board which the FE used to highlight the inter-team dependencies (sensitive information has been removed). This change brought in the most benefit according to LTM, "*[t]he one [change] that stands out is the move to the Kanban board for all the issues that were happening with Nexus. I think that's the one that provided us with the most benefit.*" The board is divided into six columns, Dependency and Integration Backlog, Blocked, In Discussion, In Progress, Acceptance Testing, and Done. Each ticket had various labels (the colour strips on the top of tickets) assigned to it, in order to make organisation and the teams who were responsible for the ticket more easily visible. Some tickets were assigned to individuals, shown by the coloured circle in the bottom right-hand corner. This board is how NIT raised, discussed, and kept track of all inter-team dependencies, increasing communication between teams by highlighting dependencies every day which involved members from various teams, and overall inter-team transparency, as indicted by SM2, "*[t]his is why we implement forums to create transparency like the Nexus Stand-Up, so that on a daily basis, at least every twenty-four hours, people can communicate and resolve dependencies.*"

The FE's whole approach to how they ran Nexus was to highlight and manage the inter-team dependencies, reaffirmed by SM2, "*[w]e then changed the approach in the stand-up and decided to focus on DIP, which is an acronym we came up with, which stand for Dependencies, Integration and Progress. In doing this, we place dependencies and integration issues in the face of every single Dev, every single day; they cannot avoid it and they don't have any lack of clarity. This made any complexities more transparent and that meant that the teams could do something about them, collaboratively. That's one way we learned from Nexus.*"

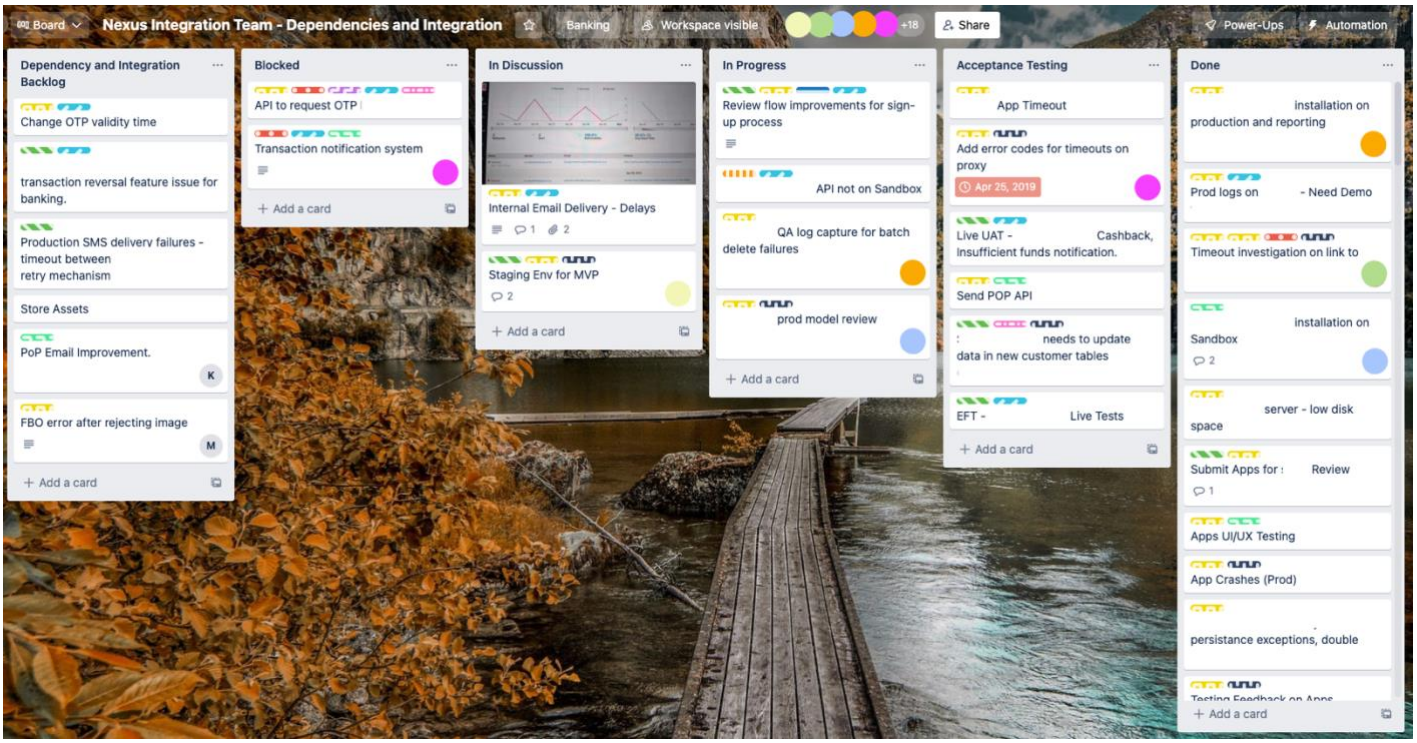


Figure 15: Trello board showing dependencies

By highlighting only the dependencies, the teams who were interdependent on each other discussed what needed to be done and the urgency of the dependency, which allowed the team who needed to do the work, to prioritise and understand the importance of that item and hence work on it with the correct sense of priority. The PDL provides further evidence to substantiate this claim: "*What has worked was getting the leads, across the other teams to communicate clearly each day and fully understand where the blockers are amongst teams, and then ensure that the team responsible for that specific item has that on their priority list. Because often what happens is that I don't know it is a priority to you, so I'll just, within my team, make this a low priority. But if I know how important this item is, because you explain to me in the Daily Nexus meeting how important it is for you; then it will be easier for us to resolve it and fix it.*". SM3 reiterates the same idea by saying: "*If person from Team A understands fully that person from Team B cannot physically build their product without this thing that Team A is responsible for, it puts a lot more emphasis on them to get that thing done. If there is a misunderstanding between what Team B needs and a lack of understanding as to when they need it for and how important it is to the implementation, nobody is really talking to me. That communication, that understanding of the teams need to talk about these scenarios all the time, this is your purpose, this is to talk to each other, find the best ways to integrate and the fastest ways to integrate; which obviously meets getting rid of your dependencies as fast as possible*." The Nexus Daily Scrum had to only focus on items which where dependencies and blockers between teams, which changed the dynamic of the project and allowed for much better clarity and

communication between teams, as well as better delivery, as blockers were addressed in a timely manner.

The FE changed the Nexus Daily Scrum into an event that was only used to highlight and discuss inter-team dependencies, also switching from a Jira board to a Trello board to address the new way of working. "*So what we changed was we were initially focusing on the details within each team, which was a waste of time because it was a repeat of what actually happened in their own Stand-Ups, which we didn't really need. So, when we changed the approach to just the dependencies, and tracked them on a Trello board, and moved away from Jira; it worked perfectly*," confirms the PDL. The LTM provides additional evidence: "*Then we switched to the Trello board, a Kanban type style was better for that because it's not really stuff for an epic necessarily, it's a little task that needs to be done or a bit of work that needs to be done and doesn't need to involve the whole team necessarily*."

6.1.4 Other components which were modified

6.1.4.1 Nexus Sprint Planning

The Nexus Sprint Planning event was attempted but was not viewed as successful and led to it being discontinued within the FE, based on various reasons, such as the inability to create an overall Nexus Sprint Goal and only the ability to give a Sprint Goal per team. Although all the teams were working on one project, each team was building a different system of the overall project, there was integration between these systems, but each team had their own objective to complete per Sprint, as described by the PDL, *"[s]o what often Nexus talks about is having a Sprint objective for Nexus. Now it was really difficult to have a Sprint objective for Nexus. We did try, and I could never have an objective, so I always gave the Scrum Master 5 or 6 objectives, an objective per team*," while adding that, "*[i]t didn't work at that level because we were building six different systems, with six different objectives and each sprint planning for each of those teams are different.*" With the inability to produce a Nexus Sprint Goal, and therefore each Scrum team having their own Sprint Goal, the Integrated Increment would not be a product of all Scrum teams, but potentially some Scrum teams where there was overlapping dependency/team integration. An example of this is the CRM system interacting with the enterprise's transaction assist system explained by the PDL, "*[l]et's take the CRM system for example, CRM said I need to do [the FE] transaction assist, then you said okay, how do we do this."* When asked for further information as to how CRM would get these requires, the PDL went on to say, *"CRM as the system has requirements, but the requirements within the CRM system, the functionality, has a dependency on another team; so that dependency is what should come back as a Nexus dependency, or whatever, a Leadership dependency or Integration*

dependency, or whatever one calls it. But you don't use Nexus to define what you're going to be doing across our environment, that's impossible." The lack of ability to give one Nexus Sprint Goal can be linked to the banking build in reality actually being various builds all related to one product, as explained by SM3, "*[i]s Nexus for you, is it actually going to work, what elements are going to work, should you actually be building separate products instead of one product and have completely different team integrate?* Further probing questions asked regarding Nexus being based off one product, many teams and banking being that with the various team products integrating into the one banking build, led to SM3 answering, "*[o]ne product, many teams. So, if you look at bank now, there are essentially 5 products that sort-of bounce off each other,"… "[t]here is integration between all of them, but one does not necessarily require the other to function, it can be built in a way in which each can function on its own for the most part. The fundamentals of Nexus are built on the, well the example they use in the Nexus guide is a security system; where literally the camera has no function without the laser trip and the motion sensors, and the motion sensors have no function without the alarm, and the alarm has no function without the doorbell with the camera on it. Where literally every single piece of the system is supposed to rely on one and other for their fundamental purpose. The analogy that comes to mind is a circus, the bank is the circus and the different elements we're building are the different shows in the circus, so it can stand on its own but it's not a circus without everything together.*"

Another factor was that theoretical Nexus assumes that there is a clear direction for the entire project with cross-functional teams. Teams who can co-ordinate amongst themselves while being able to plan all together, foreseeing dependencies on other teams, and having a vision of the entire project, but this is not always the case as explained by SM3: "*Planning does not go as smoothly as Nexus thinks. Nexus assumes that you have the vision of what you want the project to be and you have clear designations for what each team is responsible for, and that each team is fully cross-functional and doesn't need external testers or external front-end developers. It assumes teams are cross-functional and it assumes that you're building a relatively well understood, single product; and it also assumes your teams are capable of doing planning all together. Our teams were not. An example given would be that when we put all the teams in the same room, the planning does not happen; you waste 6 hours or 40% of your developer's time.*"

The amount of time spent on planning within the FE was seen as a negative due to spending too much of the team's time and not allowing for enough time within the Sprint to actually work on the development, brought to light by the PDL: "*[S]o I would have also pushed back on say, a lot of hours were wasted on Nexus Planning. So, you'll have a two week sprint; you'll have*

one full day of Nexus Planning, you'll have another two to three days of individual team planning, questions coming up, very little time for devs to dev, and then all of a sudden, you'll have demo and a retro. So, Nexus planning doesn't really need to happen, individual team planning needs to happen; dependencies that come out of your team planning needs to be addressed at the Nexus level and we monitor the rest of those dependencies."

6.1.4.2 NIT Scrum Master

The role of a SM who had the main role of ensuring the implementation of Nexus on the Nexus integration level, as it states in the theoretical framework, was not followed in the FE as all SMs were expected adopt this role. SM1 says: "*Nexus integration team, at that level there should be a dedicated Product Owner and a dedicated Scrum Master that just focus on that team. That way, they make sure that the cadence of that team is working well which filters into the cadence of the other teams*," but since there were a lot less SMs than there were teams in the FE, SM1 indicates that *"[a]t that point, I was Scrum Master of two teams and my colleague was a Scrum Master on three teams,"* this meant there was no capacity for one SM to be the NIT SM within the FE. SM3, who came into the build later, highlights the need for more SMs expressing that "*Nexus says that each team has a Scrum Master and Product Owner, while there may be a team, there is a Nexus level Scrum Master, there is not really a team Scrum Master. One person does not have the time to sit with five teams every day, I can't dedicate a full day to each team. I can't see one team for one day a week and then never again, it's not possible*."

6.1.4.3 Nexus Sprint Retrospective, Nexus Sprint Review, and Nexus Sprint Backlog

Some of the Nexus roles, events, or artefacts, stated in the theoretical Nexus framework were not implemented or fell away without much time being given to them. When the Nexus Sprint Planning was removed, its dependent events, at the Nexus level were also removed. As stated previously in this section, the Nexus Sprint Review was replaced with the individual Scrum Team Sprint Review. Thus, Nexus-level roles, events, and artefacts, except for the Nexus Daily Scrum, were replaced by Scrum-level roles, events and artefacts, as shown in a statement made by the PDL: "*I think that the whole Nexus idea of a team doesn't work, Nexus is not a team. So, a team meaning, a team of people with different skills on the team and then setting team objectives on what our objective as a team is going to be. That does not work, because the way that we have implemented Nexus is, we called it Nexus; but it was basically all the leads from the different teams, so each team has their Sprint objectives to meet, met on a daily basis to share any integration problems. Discussing integration problems and challenges between my system talking to your system, it's not a team*."

Within the FE, the Nexus-level structure disintegrated to a certain extent with the main focus being on that of the Nexus Daily Scrum and the inter-team dependency board. The Nexus Sprint Retrospective happened once, as stated by SM3: "*Retrospectives-wise, once again we don't really do them, we've done one that I know of. So practical versus theoretical on retrospectives, once again you'd have to do the Retrospectives to see the benefits of it. Do I think that if we had implemented retros more religiously, things would have gotten better? Definitely. There are a lot of issues on the Nexus integration level that needs to have been dealt with, needs to be resolved, and processes that could have been put in place to make integration considerably easier, that would have come in Retro and didn't, because they didn't exist"*. A suggestion for the way that the enterprise could have included Nexus Sprint Retrospectives, provided by the SM3, is to use the Retrospective in an ad hoc way if the project is behind schedule to detect possible causes. "*I wouldn't necessarily afford Nexus level Retros to that degree, it would be a case of if anyone has seen anything going wrong or if we are falling behind for some unknown reason. Let's call a Retro and try and figure this out. Are we deviating from our expected standards and are we losing track of where we thought we are supposed to be. If we come to that Retro and we are under the illusion that "Hey, life happens" and nothing we can fix has resulted in us being later than expected, then we carry on as normal*."

SM3 also highlights the lack of Refinement within the FE: "*One I don't fully understand that well, because I have not actually seen it in action, so it's one thing reading how it's supposed to work and seeing how it's supposed to work; product backlog refinement just purely doesn't exist. Because we 'don't have time', and yes, I say this in inverted commas, 'we don't have time'. We don't do PBR and as far as what I can understand it is a massive part of Nexus, it's a massive part of being able to map your dependencies. Why we don't do it, the general feedback I have got is that we just don't have time; which is fair enough as we don't have time to spare. The theoretical knowledge that I have, seems to sort-of underpin that you must do PBR in order to have a successful Nexus integration; you have to be able to map your dependencies, etc. This I have not seen practically implemented, so I cannot tell you if we're doing it better, or if the outcome has been more successful on PBR or not, but I can just tell you that we do not do this."* Through observation, the only aspect of Refinement that could be considered was done by the PO (sometimes assisted by the LTM), who managed the Product Backlog.

6.2 Unmodified theoretical Nexus roles, events, and artefacts

Some components, i.e. *roles*, *events* and *artefacts*, from the theoretical Nexus framework were adopted without any adaptations within the FE. The unaltered components were not were not afforded considerable focus during the interviews, as these were just considered the

implementation of the project, incorporated from Nexus, but can be accounted for, based on observations, and some extracts from the transcribed interviews.

Regarding the *roles*, the Development Team and Nexus Integration Team (NIT) were not significantly altered or changed on purpose, but may have differed from the exact textbook definition, due to the FE's structure. The Development Team remained as prescribed in the framework. Individual Scrum teams within Nexus and the Nexus Integration Team consisted of the PO, SMs, BAs and Team Leads. The framework does state that the NIT may consist of persons who are not part of the Scrum teams to gain from their specific area of knowledge, but the enterprise did not include anyone from outside of the Scrum teams within the NIT.

Regarding *events*, Sprints, Sprint Planning, Daily Scrum and Sprint Retrospective were all implemented according to the framework prescriptions, since the enterprise already implemented Scrum before they experimented with Nexus. Using Scrum previously aided in the transition into Nexus since team members had less resistance to change, indicated by SM2: "*[T]hat's one thing; the other thing is, because Scrum was already implemented successfully, it was easier for the teams to understand Nexus, it was a very small learning curve. John Gold, a systems expert, he wrote that: "A complex system that works almost invariably evolves from a simple system that worked." Meaning, start small, build a foundation, and then scale on top of that. If you are going to build a complex system from the get-go, you are going to struggle and it's going to take you a lot longer than you planned*." The PDL reiterates this sentiment: "*I think the adoption was fine, people understood what needed to happen and they did what they needed to within their individual teams and within the Scrum.*" Sprint Planning and Sprint Retrospectives are mentioned in a quote from the previous section, as well as by SM1, "*[w]e did Sprint Planning at the beginning of Sprint, ending Sprints with Reviews and Retrospectives and then starting again; when you get that cadence going very well, then it works well. Not only on the Nexus level but in the teams,"* but from observation it could be seen that Sprint Retrospectives were championed by the BAs and/or SMs, with little buy-in from management, unless serious internal issues emerged.

Regarding *artefacts*, participant observation indicates that the Product Backlog and the Sprint Backlog were not altered from the theoretical framework. The PO managed the Product Backlog, with the assistance of the LTM in various aspects such as IT infrastructure. The Sprint Backlogs per team were discussed between the PO, the teams' SM and/or BA who led the Scrum team, which determined the multiple Product Increments that were produced per Sprint.

6.3 Findings about Nexus within the enterprise

There were other learnings about Nexus, regarding the framework in the practical case study environment which were identified based on the implementation, execution, and the dynamic within the enterprise.

The SMs expressed that they felt the need to be more stringent on the implementation of certain components, SM1 said: "*Do your planning. On a Nexus integration level, do the planning; take your time. Get it on a board, user-map it, take string and red pens and make the dependencies visible; do the planning. Otherwise, there is no visibility, your visibility gets even more murky. If you don't do the planning, then the teams don't know where they are at. When you get to the actual Nexus-Sprint Planning, then you do your Refinement. Walk out of the room with a goal, an attainable goal for a sprint. We were not strict enough on implementing those basics,*" while SM3 spoke about the Nexus Sprint Retrospective by stating: "*I'd also, oddly enough be stricter about having certain things, like Retros; but it wouldn't be a case of you must Nexus level Retro because you are doing Nexus, it would be a case of, we're doing team Retros because of what Scrum says. Scrum says we do Retro because sometimes you just need to either have a team-build or there are major issues you need to deal with.*" This was a result of another issue regarding the implementing of the framework, which was a lack of Nexus experience, as indicated by SM3: "*I think the fact that we had inexperienced Scrum Masters and Product Owners*," adding prescriptive measures: "*if you're going to do Nexus, and you're going as we did, as this company did; we went in as very inexperienced, nobody involved in the Nexus implementation actually, practically had done Nexus before.*"

The lack of experience induced implementation issues and insufficient coaching, leading to a lack of management buy-in, indicated by the PDL: "*I think the issues we had weren't really a problem with the framework, I think it was more the understanding and the implementation that is actually realised in our organisation*," who continued with "*I think the Scrum Masters spent too much of time analysing the theory and implement theoretical components of what Nexus spoke about; in comparison to saying, 'Here's guidance, which they understand, and let's make it work for our organisation practically*'." SM1 states: "*There isn't really anyone to blame, it's more of the fact that the coaching was not done with the Product Owner; I wasn't the lead on the product, so I didn't have the opportunity to do that coaching. I think that a person needs to understand that if the Product Owner and Scrum Master work well together then everyone will fall in underneath. If there is a push-and-pull the whole time between Business and Tech then Tech will always lose.*"

The lack of management buy-in created a strain between SMs and PO, highlighted by SM1, "*I would have approached my relationship with the Product Owner and stakeholders differently*

in the beginning and let them understand the buy-in, let them understand the value of it. Perhaps they didn't understand the value of the process we were implementing, maybe coaching wasn't executed properly and that's why it's never been supported," "*I would have loved to have seen a Product Owner-Scrum Master relationship. This relationship would have had the two standing as a united front because often times there was no unity. The Product Owner should have been able to understand the pressure, but still made time for the retrospectives and planning. So those type of things that fell by the wayside were because there wasn't a unity in leading the teams.*" SM1 felt that this break in relationship led to some of the components being removed, "*[t]here is always a push and pull between the Scrum Master and Product Owner, most time of which the Scrum Master lost and when that happened, we didn't have Retrospectives.*"

Another finding was the importance of getting all team members on board with the framework and process, according to the SBA, "*[w]hat I have learnt with the Compliance group was that you have to get your team organized, you have to give them this view of what we are building towards; because I felt like so many weeks were wasted where everybody was so confused just trying to find their feet, not really know what it is we are building towards. Anybody buying into any idea, must be 100%, else it won't work; so it doesn't help you come up with an idea like Nexus when half of the team is kind-of in kind-of not, 'cause you have to have dedication, you have to have people that attend the meetings that follow up on what they are supposed to be doing and do what they are supposed to be doing*." An overall view of the end-product allows for better team buy-in, in turn, increasing team member participation. Giving team members the clarity of a build and a sense of direction minimises confusion and leads to the ability to be more agile in their approach, as indicated by SM2: "*I think the biggest thing we learnt, was that in order to be nimbler you need to give everyone in every single team shared consciousness. So, every person needs to understand the entire system, to an extent, they don't have to be an expert with complete knowledge about something, but they need to have deep knowledge about their domain and then breadth of knowledge across other domains*."

6.4 Non-Nexus related findings within the enterprise context

Some of the learnings from the case study became evident from the general setup and running of the project but were not directly linked to Nexus.

6.4.1 Globally distributed teams and team groupings

One of the main issues was the global distribution of teams and how the teams were grouped. Ideal circumstances for increased communication and collaboration are teams that are collocated, as highlighted by SM3: "*[S]o however much communication you can manage to get*

through Slack and through Skype, and through regular phone calls throughout the day; having someone or the teams you are dependent on, literally sitting next to you, is the most efficient way to get work done," who also mentioned that, "*[w]hat has worked is having the teams collocated and having all of your teams which are dependent on one another and require cross-functionality between teams, to sit at the same site; at the same time. This hasn't worked as one of our major dependency teams is on a different continent*."

Other than collocated teams, another aspect mentioned previously is how teams themselves are created. Ideally, as stated by SM1, members should choose where they would like to work, something the FE had not done, "*[t]hey also based it on Tech Lead skill in current teams, they based which team was best suited to which area based on what they are currently doing. We did give the people an opportunity, just after launching, to move and we had two movers that said they don't want to be where they are, and they were moved. It wasn't ideal though, I would've liked everybody to choose their own teams, I would have preferred that*." Team selection was presented as an option later, after teams were grouped. However, teams were, and for the most part, remained based in a single location. SM2 states: "*[w]e could have said to the people to self-organise themselves into efficient teams, let's look at the functions we need to build and let's make sure we organise ourselves into teams to reduce dependencies. So, we shouldn't have looked at it in a way where one team is in India, therefore they work on this one function; this team is in Mauritius, they work on one function; this team is in South Africa, they work on this one function. We should have eliminated the function and geographic side of things and rather looked at what features there are and let teams, based on their skills, form around the feature to eliminate dependencies and the constant back-and-forth.*" Having teams based by location, did affect inter-team dependencies, not only from a collocation and communication perspective but also other factors, such as time zones, this is highlighted by SM3: "*Okay, I'm going to call them 'the dependency', the team that we are dependent on is 5/6 hours ahead of us. By the time we are hitting our peak delivery and performance here around lunch time, they are getting ready to go home. When we need them, when we're in the midst of fixing issues, finding bugs, and resolving things, they are now not available to assist.*"

There is however a trade-off when structuring globally distributed teams either based on location or by skillset, namely, the need for a single team to communicate and function with the challenges of being globally distributed versus the decrease in cross-team dependency, as mentioned by SM2, "*[a]gain, if teams weren't formed based on geographical location. That's a big one because we chose to go with teams centred around their geographical location, we created more dependencies. If we could have helped the teams self-organise into teams around features, according to skill sets of employees regardless of location across the globe we would*

have reduced dependencies. The trade-off there would have been that we had geographically distributed teams which needed to dial in one another to have meetings and stuff, that is an operational overhead, so there was a trade-off there".

Another factor to consider with team groupings is the number of members in a team, as raised by the LTM: "*The other problem that we have is we don't have enough people to have just one team, maybe four to five people, responsible for just one set of web services. There is always ten or twelve people who do six different things."* The more work each team member is responsible for, the easier it is for work quality to be negatively impacted.

6.4.2 Technologies used within the project

The choice of technologies to use within the project, whether for tracking or as the general means of communication, is critical to the functioning and management of the project, especially when large numbers of people are involved.

The FE brought in new technology while simultaneously introducing Nexus which was problematic, as explained by SM1: "*Do not use new technologies. Ever. What bit us badly was, if we had the technologies we always had, that we knew and that we worked with; and we had started Nexus on top of it, we would have delivered far earlier. I'm sure of it. Also, we did a new introduction of scaling and a new introduction of new technologies, what took really long was understanding the technologies and then building the bank.*,"… "*if you are going to do one or the other, put the new technologies in place and then tell the leaders that it is going to take three months to get used of the technologies; make hard decisions fast. We didn't do this, we waited nine months to ask whether the technology was working for us, and when one didn't, we kept it in the mix because it was too late in the game to change anything. Now we are working around it, trying to make it fit. We are forcing a square peg into a round hole. So, don't do that, don't try to introduce new technologies and scale at the same time. Just do scale.*" Trying to introduce too many changes at the same time does not allow for understanding and adaptation of one change fully, before moving onto the next. This leads to issues with these changes only being realised late. It is also easier to implement and manage smaller changes done incrementally, than all done at the same time. The LTM links the matter of tools of collaboration and communication together by stating: "*I mean the idea with Nexus is that everyone knows everything that's going on and there is transparency to what's happening in the teams and stuff like that; I don't think that's quite there yet but in terms of the transparency within the team, if you go into each team's Jira board you can see what they are doing, sort of, because the tickets that they put on usually only mean something to the team or just to the developer involved. It*

would be a one-line message on the ticket and then nothing in the description, so if someone else wants to take over, they won't know what the ticket means."

The FE also had not standardised channels of communication, causing missed communication and a sense of general confusion. This is exemplified by the SBA and SM3 respectively: "*[M]aybe the tools of collaboration, I still feel like things get a bit lost in all the channels we're using*," and "*at the moment, there are 15 different channels of communication for the same thing, the current Scrum Master, being myself, is half the time unsure of which is the head and which are the toes of the conversation; because things are just completely all over the place. If anyone was to join, a developer or tester, the banking project at this stage, they would not know where to begin; they wouldn't be able to catch-up because there is no agreement on the best approach to take.*"

6.4.3 General 'big build' issues

More general aspects that come with most big projects, can be classified as 'stereotypical items', referred to be SM3: "*Stereotypical items which would come up from any post-mortem of a project; which are communication, understanding specs, getting clear business requirements; which goes back to getting your stakeholders involved. These are the things that go with basically any big project, you're going to have communication issues, clarity of spec issues and in this situation, you're going to have integration issues.*" These are items that were noticed by some of the members, and the PDL highlights the importance of clear business requirements: "*My main lesson is, ensuring that across the different systems, our specs are fully locked down. Business Specs, are fully locked down, the timeframes which are set to actually build this is fully understood by all parties, adequate time given for regressions testing; across the board.*" The SBA explains the negative effects of incoherent timelines, "*[t]here is still lapse in communication regarding a lot of things, for example the dates, when are we going live? There are four different dates, nobody knows what the real dates are. There are still things being built somewhere, that people are not aware.*"

The SBA further highlighted the necessity for clear communication and the risk of assumption: "*I feel like people assume a lot and think that's in Nexus. I feel like everybody still does a lot more of their own thing to get to their Point A or Point B, and it's understandable when you're under this much pressure that you don't collaborate as you should, maybe go into your own little tunnel vision just to get things out. I think people communicating, or relaying a message, doesn't always get communicated correctly to the person getting the instructions. I found that a lot with myself. If I ask someone how is something working, they'll answer me and that is how I'll implement it; then when I show it to them, they are like 'we are not going to do that now'*

or 'maybe that's not the best way to build it', they always think that if you tell me this, then that is what I'm building now."

Lastly, remember the human factor, as SM3 indicates: "*One thing I do think, in specifically team Retrospectives, even if it's fifteen minutes; in those fifteen minutes you agree to go and have a beer after work. The human side of things. There is a general requirement to get things out as fast as possible and Nexus or scale scrum are designed to address this as lean and efficient as possible, but the flip side of the coin is that people get seriously drained, they get worked to the bone. I think that in all the excitement and pressure, we lost sight of the fact that while business is impressed, we are pushing the development teams to the point that they can't breathe to get to market faster. Nexus and scale Scrum, if executed properly, gives us the opportunity to take fifteen minutes to give back to your people, to take care of your people. To tell them that they are leading the trend in this industry and you deserve to be rewarded for this, even if it's donuts at lunch. That human aspect, I believe, very often gets forgotten, not deliberately, but it does get forgotten.*" On big projects it is easy to forget and not celebrate the small wins, to only focus on the main goals as there is pressure to complete a huge project. People are not machines. They need a break, balance, and appreciation for their efforts.

7. Synthesis of insights

Changes between theoretical and practically-implemented Nexus, as well as the reasons for these changes are synthesized in section 7.1, accompanied by potential solutions and observations of these factors. Section 7.2 provides insights into general learnings found regarding the Nexus framework, while section 7.3 focuses on insights which are *not* related to the Nexus framework but overall findings within the case study.

7.1 Theoretical versus practical Nexus insights

Table 11 shows a summary of Chapter 5 and Chapter 6, displaying a visual of which concepts were changed between theoretical and practically implemented Nexus, as well as the reasons for these changes in the case study. Table 11 shows the concepts which either were *not* changed, or were *not* implemented, giving a more complete view of the differences between the theoretical textbook version of Nexus and the implementation within the FE. The data gathering and the reasoning given for these differences have already been given. The purpose of this section is to highlight the insights obtained from this case study when implementing Nexus at the FE.

7.1.1 Nexus Sprint Review insight

The Nexus Sprint Review failed due to its format. Yet, the Nexus Sprint Review was only attempted once and no corrective action could be implemented with a follow-up attempt. Potentially smaller Reviews between teams whose systems actually interacted with each other, where inter-team dependencies have been worked on, would have been better than having the presence of all teams in a single Review. The large-scale agile development project includes many systems and members, and it is difficult for every member to understand design details from an overarching perspective. The case study indicated that it is better to keep the NIT team responsible for managing the complete overarching view and have inter-team Reviews where there is an overlap of work. Inter-team Reviews would shorten meeting times, since less content is reviewed, as well as increasing focus, because members would have worked on what was demonstrated, allowing teams to see how their work has integrated with other teams and showing the importance of their work. Inter-team Reviews would increase the number of Reviews presented in Nexus but decrease the number of the Reviews in the FE, as the enterprise resorted to individual Scrum Team Reviews placing a lot of time pressure on the PO who needed to attend all of these Reviews. Stakeholders and end-users should also be included in these Reviews as their feedback is needed to make smaller and needed changes.

Table 11: Theoretical versus practical Nexus implementation

Component	Textbook	Practical	Reasoning for difference
Roles			
Product Owner	Responsibility of boosting product's value and responsible for Product Backlog.	Main Product Owner and then Business Analysts became POs for their systems, in communication with main PO.	Managing more than 50 members and various teams is overwhelming and cannot be done easily. Having the BAs take over this role within their system build allowed for pressure to be taken off the PO and a single point of contact between the PO and that system build.
Scrum Master	A Scrum Master of NIT and of individual Scrum Teams.	There were originally 2, later 3, Scrum Masters, and more than 6 teams. There was no 1:1 relationship. No Scrum Master of NIT, all Scrum Masters adopted this role.	Lack of Scrum Masters within the enterprise.
Development Team	-	-	N/A
Nexus Integration Team	The framework does state that members from outside the Scrum can join the NIT to provide expertise.	The enterprise only used members from within the Scrum.	There is no discussion on this topic and no clarity regarding if it was even considered at the FE.
Events			
Sprint	-	-	N/A
Sprint Planning	-	-	N/A
Daily Scrum	-	-	N/A
Sprint Review	The Sprint Review should be replaced by the Nexus Sprint Review and should not be part of Nexus.	The Sprint Review as brought back and the Nexus Sprint Review was removed	The Nexus Sprint Review failed so each individual Scrum team had a 'demo' at the end of their Sprint. BAs, SMs, PO and sometimes external stakeholders would attend.
Sprint Retrospective	-	-	This was implemented by the Scrum Masters or the Business Analysts. Little attention was given to this event from management unless a big problem was raised, where feedback was given directly to the PO.

Component	Textbook	Practical	Reasoning for difference
Nexus Sprint Planning	Should be like Sprint Planning but on a Nexus level where the refined Nexus Product Backlog is used to plan next items to be taken into the next Sprint.	Was removed from the FE's Nexus structure.	PO could not give one goal for the Nexus level sprint (Nexus Sprint Goal), could only give a Sprint Goal per team.
Nexus Daily Scrum	Members from different Scrum groups gather together to discuss integration concerns and inter-team dependencies. What was done with the team the previous day and adjustments to plans, backlog updating, etc.	Started as a normal Daily Scrum, each person saying what was done, to be done, what were the blockers. This changed to only looking at inter-team dependencies.	With each team giving updates on what they had done, the inter-team dependencies were neglected. The FE moved from Jira to a Kanban board on Trello for the Nexus Daily Scrum and highlighted only inter-team dependencies, resulting in shorter meetings and better communication between teams as well as the urgency of the dependencies among the teams.
Nexus Sprint Review	This event should replace the individual Scrum Team Sprint Review with all team members attending, including other stakeholders. This event is done at the end of the Nexus Sprint and allows feedback on the complete Integrated Increment.	This event was attempted once and then was taken out of the FE's Nexus structure.	The meeting went on for hours, so focus was lost. Not all teams' deliverables integrated with all the other teams and attention was missing. Team members not based in South Africa had difficulty in engaging with the meeting.
Nexus Sprint Retrospective	This event ends the Nexus Sprint and is a time to deliberate on what went well in the Sprint and what needs to be changed in the next Sprint. This event includes the NIT and other interested persons, discussing inter-team dependencies and issues, as well as using information from the individual Scrum Team Sprint Retrospectives to highlight predominant issues.	This event was attempted once and then removed from the FE's Nexus structure.	Not much was mentioned in the data gathering on the one event that occurred, except for the fact that only one had occurred.
Refinement	Refinement is a mandatory event where teams work with each other to decompose the Product Backlog into smaller, more manageable tasks, giving the ability to understand inter-team dependencies.	The event did not happen on a cross-team basis. Rather, the PO managed the Product Backlog, sometimes together with the LTM, and then discussed those items that would be brought into the next Sprints with individuals that led the teams, such as BAs and SMs .	The reason given for not having this in the enterprise was a lack of time.

Component	Textbook	Practical	Reasoning for difference
Artefacts			
Product Backlog	-	-	N/A
Sprint Backlog	-	-	N/A
Integrated Increment	In theory, this artefact is described as a product of integrated work done by all the Scrum teams within Nexus.	Some Scrum teams within Nexus had overlapping dependencies or team integrations where their work would be combined as an integrated component, but some teams produced increments that were system specific and did not integrate with other systems	Each individual Scrum team had their own Sprint Goal, as there was no Nexus Sprint Goal, so each team worked towards creating their own incremental deliverable which potentially had some integration with some of the other teams within Nexus.
Nexus Sprint Backlog	This artefact is used to understand inter-team dependencies and emphasise these items, altering them daily, based on progress presented in the Nexus Daily Scrum.	Within the FE, this artefact was not really considered. As explained in this section, a dependency board was used in the Nexus Daily Scrum that served as the Nexus Sprint Backlog.	Without the Nexus level planning, there was no perceived Nexus level Sprint Backlog, but rather a change of the Nexus Daily Scrum event to highlight only inter-team dependencies, also creating a Nexus Sprint Backlog.

7.1.2 Nexus Daily Scrum insight

As mentioned, in Chapter 6, once it was fine-tuned to only highlight the inter-team dependencies, the Nexus Daily Scrum was one of the most successful events of Nexus in this case study. The reason for its success is that teams started communicating to other teams and were able to provide a sense of priority to other teams about existing dependencies and their importance. It created transparency between teams by increasing communication and collaboration. This Nexus-adaptation only came into practice a few months into the project and some of the feedback indicated that had this change been implemented faster, the results would have been better and inter-team dependencies would have been noted a lot quicker.

7.1.3 Nexus Sprint Backlog insight

Considering the theoretical Nexus Sprint Backlog, this is the suitable place where inter-team dependencies are managed and reviewed in the Nexus Daily Scrum. Examining the structure and the way that Nexus was perceived in the FE, the theoretical Nexus Sprint Backlog was excluded, but considering the theory behind this artefact as well at how the FE implemented their Nexus Daily Scrum, the FE was actually, potentially by coincidence, using a Nexus Sprint Backlog by introducing the dependency-highlighting Trello board. A board which shows only items considered as dependencies, not items that each team is working on, and managed and altered in the Nexus Daily Scrum re-defined the artefact Nexus Sprint Backlog at the FE.

7.1.4 Nexus Sprint Planning insight

The inability to define a Nexus Sprint Goal, led to the failure of the Nexus Sprint Planning event, and inadvertently led to the failure of some of the other Nexus-specific components. Proper Nexus Sprint Planning would have allowed for dependencies to be factored in earlier, allowing teams to include these in individual Scrum Team Sprint Planning, rather than on an ad-hoc basis from what was raised in the Nexus Daily Scrum. Although all team members were working towards contributing to the one project, each team was working on a different system, systems that would need interfaces, but completely separate in terms of their function. The separate systems in itself could have been an indicator that, although each system was needed within the build to be utilised in the banking project, potentially these systems could have been viewed more as mini individual projects and Nexus could have been approached differently. Within the banking build, core teams existed and most systems relied on these, such as the Web Services/API team; but other systems, such as CRM (Customer Relationship Management) and Compliance systems, both integral systems within the banking platform, were not linked and were not directly dependent on each other. A customer had to be approved using the

Compliance system, before the customer could be signed up as a banking customer. Once a customer has been approved as a registered bank customer, the customer would be assisted via the CRM system, but the functioning of one system is not directly linked to the other.

7.1.5 Nexus Sprint Retrospective insight

Nexus Sprint Retrospectives were not very widely discussed in the data gathering process, only that it happened once and was removed for the FE-adapted Nexus structure. The Nexus Sprint Retrospectives is meant to evaluate the effectiveness of the Sprint and provide suggestions for improvement, extracting feedback from the individual Scrum Team Sprint Retrospectives on major issues. As stated previously, the Scrum Team Sprint Retrospective was not viewed as a priority by the PO and management and should any issue have arisen it was just raised directly to the PO's attention. A way to have potentially utilised the Nexus Sprint Retrospective mentioned in Chapter 6, was to have this event only when the project was behind schedule to understand what the cause. Using this approach allows for flexibility regarding the Nexus Sprint Retrospectives but also maintains the ability and need to discuss key issues at a NIT level.

7.2 Insights surrounding the general use of Nexus

The implementation and use of Nexus within an enterprise is highly dependent on the dynamic of the enterprise itself. The FE had members with no prior experience in Nexus attempting to implement this solution. This led to weak advocacy of several the Nexus events as well as a lack of coaching to the FE management on Nexus itself. The lack of coaching leaves gaps of knowledge and understanding, and an overall paucity of management buy-in, which is detrimental to the success of an agile solution. Within the FE, there was also friction between the PO and SMs, which should in theory be a unified team, this will cause a breakdown in the agile solution as these are both main members of Nexus who are not in agreement.

Team buy-in and overview understanding, on a high level, is needed for the solution's success. With a large scale project it cannot be expected that every member is fully aware of the entire build in detail, but team members need to have a high level understanding of the fully project and how their system/piece of work fits in. Understanding of their work in the main build, along with how the agile solution brings the build together, will assist in better solutions being developed and increased team buy-in which will contribute positively to team understanding and participation in the agile solution.

7.3 Insights into non-Nexus related learnings

There were valuable learnings which the case study produced that were not directly linked to Nexus specifically, these have been included to give further value to the dissertation and offer

more insight for either those with a similar setup/project, or to research scholars considering further studies regarding large-scale project builds.

For optimum communication and collaboration between teams, teams should be collocated. Unfortunately, with big builds and enterprises that have teams dispersed globally this is not always possible, as is the case in the FE. Globally distributed teams brings in its own complexities such as lack of communication and understanding, as well as how to setup the teams. Teams can be setup based on skillsets or based on location, the FE chose to base teams on the latter, this in itself has pros and cons. The advantage is that the teams themselves are collocated but the disadvantage is an increase in dependencies between teams, which did not only present distance as a problem, but also other problems such as differences in time zones. The insight regarding globally distributed teams and how to group them is that there is a trade-off between location versus skillset, and this is something that each enterprise would need to consider and base on the enterprise specific dynamic. The number of members in a team is also something that needs to be considered carefully, team members working on multiple items, or having extreme split-focus by doing a multitude of items, can lead to burn-out and decreased quality of work.

When the FE implemented Nexus, it also implemented new technologies to assist with the transitioning and the implementation of Nexus. Trying to implement a scaling solution as well as a new technology proved to be a challenge that the FE, in hindsight, this should have been avoided. The insight derived is that adjusting and changing too many aspects should be avoided. Rather compartmentalise different aspects and introduce them in a staggered approach. Regarding technology used, another insight was that of the chosen channels of communication. If there is no standardised tool and channel of communication, communication and messages are lost amongst the various platforms, creating confusion and hindering the build. The insight gained is that an enterprise should find a single channel that works best and stick to that, ensuring that everyone is on the same level of understanding and less communication is lost in the process.

Having clear business specs, and the ability to set timeframes is key to avoiding confusion and ensuring the teams' understanding is congruent. Standards need to be implemented, not only with the tools being used, but how they are used in order to ensure better communication, easier management and collaboration, and increased shared understanding.

8. Conclusion and Recommendations

Scaled agile solutions have become more popular in recent years to obtain the benefits that agile offers, but using agile practices on a much larger scale than originally intended for. Big projects have various challenges due to the magnitude of their builds, with different challenges for different types of projects, environmental settings, and enterprise dynamics to name a few. The different contexts need to be considered and catered for in large-scale agile developments.

One of the challenges highlighted in this case study was the lack of *inter-team dependency* considerations when scaling agile, and leading to the paucity of inter-team communication and collaboration, which in turn leads to delays in builds as inter-dependencies are unidentified and become impediments for progress in the project. The study considered Nexus, a scaled Scrum framework, which is a framework mentioned and referred to in existing research studies, but with limited empirical research, such as Nexus-specific case studies. This study contributes to addressing the current literature gap regarding Nexus, comparing a practical application of Nexus against the theoretical framework, with a focus on improving *knowledge regarding inter-team dependencies* within a large-scale agile project.

In this chapter, section 8.1 discusses the research questions posed in Chapter 1, which are revisited and answered, bringing the focus back to the central theme of the study. Section 8.2 summarises and discusses the key findings of the case study, ending this chapter with section 8.3, which gives recommendations of future research topics.

8.1 Research questions

Beginning with the secondary research questions (SRQ) and building up to the primary research question (PRQ), section 8.1 presents answers to the research questions of this study.

8.1.1 SRQ 1: What issues were experienced at the enterprise with regards to their use of Scrum?

With the originally-established Scrum solution within the FE, the team members were working well within the individual teams but were working in silos. Working in silos resulted in the teams gaining cadence within the team, but not knowing what other teams were working on and therefore no out-of-team-scope considerations were considered or taken into account for planning. Obstacles emerged as a result for the project, as external-to-the-team issues were not investigated and resolved. Using the Five Whys method, the root cause for inefficiencies across teams was discovered to be a lack of transparency between teams as Scrum was implemented on a sub-system or individual-team level.

8.1.2 SRQ 2: Who/what is most affected by the identified issues?

The most affected aspect of the project build was knowledge of inter-team dependencies, or lack thereof. Using a causal-loop-diagram, communication amongst teams, collaboration amongst teams, and knowledge of inter-team dependencies were causally linked. With the teams working in silos, there was little to no communication between teams and therefore, a lack of collaboration, leading to insufficient knowledge of inter-team dependencies. The fact that the scale of the project was the biggest that the FE had implemented in the preceding years made it clear that inter-team dependencies had not been accommodated and with globally-distributed teams, this was a chief concern, as all teams were working on system builds linked to one primary project, namely the digital banking platform project.

8.1.3 SRQ 3: Does the problem instance feature as a class-of-problems in existing literature?

Within literature, numerous academic sources highlight the need for using agile practices in large-scale projects, whereas *agile* as a software development approach, was designed for small-scale projects. Previous research indicates that with larger projects, various issues exist, including challenges of managing multiple members, loss of visibility of the product development, insufficient requirements, geographic distribution challenges, limited coordination amongst teams, and inconsistency of coding standards. Coordination among teams relates to the problem of communication and collaboration between teams on a large scale project, which is directly linked to *knowledge of inter-team dependencies*.

8.1.4 SRQ 4: What knowledge areas could be useful to address the class-of-problems?

A literature review and developing a codebook to analyse existing literature, led to themes becoming evident in order to identify and understand solutions offered by literature regarding *scaled agile solutions*. Although many scaled agile frameworks exist, to assist with the utilisation of agile methodologies for large-scale developments, this study highlighted four frameworks: SoS, SAFe, DAD, LeSS, and Nexus.

8.1.5 SRQ 5: What was learnt, in literature, from the implementation of scaled agile frameworks?

Through analysis, SAFe, DAD, LeSS and Nexus were selected to be fully evaluated and compared. Section 2.3 provides the full scaled agile framework evaluation results, presenting pros and cons for each framework. The key findings indicated that SAFe, although being a popular scaled agile framework, is extremely complex due its comprehensive nature, whereas LeSS is less complex, but has minimal structure and relies on concepts that the FE is already

struggling with, such as mindset and communication. DAD included delivery in its scope, which is more comprehensive than Scrum, but its implementation creates confusion, as some features lack detail. DAD, similar to LeSS, is classified as a medium-complexity framework.

Nexus is a lightweight framework and an extension of Scrum, but there is still a lack of empirical knowledge regarding this framework. Consequently, all its implementation flaws are not yet known. Through evaluation, LeSS and Nexus were selected as the best solutions to be compared because they both were based on Scrum and the FE already used Scrum. Nexus had the advantage of reducing the risk of resistance to change, an issue that has been raised with scaled agile approaches. LeSS ranked higher than Nexus on inter-team communication, but LeSS is more customer-centric whereas Nexus is more inter-team-collaboration orientated. As inter-team collaboration is a main problem area within the FE, and LeSS relies on concepts which the FE is lacking in, Nexus was chosen as the best suited scaled agile solution for the enterprise.

8.1.6 SRQ 6: What are the key components which can be compared to evaluate the differences in the practical and theoretical frameworks?

As Nexus is an extension of Scrum, it uses Scrum as a baseline for adding and removing or adapting components to create a scaled agile framework. Scrum contains three roles (Product Owner, Scrum Master, and Development Team), five events (the Sprint, Sprint Planning, Daily Scrum, the Sprint Review, and the Sprint Retrospective), and three artefacts (the Product Backlog, the Sprint Backlog, and the Product Increment).

Nexus removed individual Scrum Team Sprint Reviews, adding one role, five events and one artefact. The key components of Nexus, along with those from Scrum, were used to evaluate the differences in the practical and theoretical frameworks. The additional Nexus role, is the Nexus Integration Team (NIT) role, the five events are the Nexus Sprint Planning, Nexus Daily Scrum, Nexus Sprint Review, Nexus Sprint Retrospective, and Refinement, and one artefact is the Nexus Sprint Backlog.

8.1.7 SRQ 7: What were the reasons that the theoretical key components did or did not work within this enterprise's environment?

In terms of *roles*, the practical PO role was the central role as indicated in the Nexus theory, but the practical application indicated that the role is overwhelming and unrealistic, since one role was responsible for managing so many members. The practical solution partially transferred some of the PO's responsibility to multiple BAs, where the BA in some teams, adopted a pseudo-PO role in the teams they were "leading". The solution eased the pressure off the PO

and allowed for a single point of contact that the PO could rely on to understand the progress of a sub-system and co-managing the product backlog for that team.

The FE did not have an NIT SM, as a limited number of SMs existed within the FE. Each SM adopted the role of the NIT SM, ensuring that Nexus was implemented within the project build. The NIT consisted, as the theory had indicates, of a PO, SMs, BAs, LTM and Technical Leads. The theory does state that members from outside the Scrum team could join the project team, but this suggestion was not implemented by the FE. No specified reason existed for excluding members from outside. The Development Team remained unchanged between theoretical and practical Nexus.

Regarding *events*, events from theory that were retained in the practical implementation included the Sprint, Sprint Planning, Daily Scrum, and Sprint Retrospective. The FE already used Scrum on an individual team level and this cadence continued. The individual Scrum team Sprint Retrospectives were not prioritised by management and any feedback on major issues were reported to the PO, which in theory should have been used to be reported during the Nexus Sprint Retrospective. Yet, the Nexus Sprint Retrospective was attempted only once within the FE and then not again. No substantial reason was given as to why the Nexus Sprint Retrospective was discontinued within the FE, although the PO did mention that the event was time consuming and wasteful. The Nexus Sprint Review was also an event that occurred only once during the practical implementation of Nexus at the FE due to its extended duration, with lack of concentration and focus, also due to the fact that not all teams integrated with the other teams. Those who were not in South Africa (where the meeting was taking place) battled to engage during the event, which also contributed to a lack of concentration and success. The fact that work presented during this event related to the Integrated Increment, but had only been worked on by some teams, also differed from the theoretical intent of the meeting. Some of the FE's teams' work did integrate, others did not, due to the inability to define a single Nexus Sprint Goal but only give individual Scrum team Sprint Goals. The Nexus Sprint Planning was consequently removed. Since there was no planning on a Nexus level due to there being no Nexus level Sprint Goal, items associated with the next team Sprint Backlog, were transferred from the PO directly to those who were managing each of the teams, BAs, SMs, and Technical Leads. The PO, along with the LTM, managing the refinement of the Product Backlog, thus differed from the theoretical intent of being a cross-team event. The PO managed the Product Backlog and therefore did the Refinement, i.e. it was not performed on a Nexus level in the FE, due to a perceived lack of time.

The Nexus event with the most success was the Nexus Daily Scrum, which was altered to only highlight inter-team dependencies and not provide updates on the teams' progress. The

modification from the theory was made to shorten the meeting time as well as ensure that the dependencies were not lost in all the other information given, this also gave those with dependencies a chance to discuss the importance of the work and allowed the team who needed to pick up the work to understand what was being blocked from this work not being done. Using a Kanban board as the Nexus Sprint Backlog, although at the time it was not recognized as this due to the lack of Nexus Sprint Planning and raising this up every Nexus Daily Scrum allowed for teams to raise, understand, discuss and give feedback on inter-team dependencies.

8.1.8 PRQ: How can the theoretical Nexus framework be adapted in order to make it a practically viable solution to improve knowledge on inter-team dependencies within a Fintech enterprise?

Answering the secondary research question and comparing what the theoretical Nexus framework dictates versus what was implemented in practice, the main contribution towards a practically viable solution to improve knowledge on inter-team dependencies, is the use of a Nexus Sprint Backlog in the form of an inter-team dependency board that is presented and discussed daily in the Nexus Daily Scrum. Teams needed to raise and discuss dependencies that impeded the project's progress. When dependencies were raised, communicated, and worked on, knowledge about inter-team dependencies increased.

8.2 Discussion

As stated in the previous section, the use of the Nexus Sprint Backlog as an inter-team dependency Kanban board was discussed daily in the Nexus Daily Scrum, and was a major success for the FE, increasing transparency between teams and destroying silos to contribute positively towards collaboration and communication amongst the teams.

It is difficult to understand to the full extent of implementing and applying Nexus, as many of the events were removed from the FE-specific implementation. Reviewing the way Nexus was ultimately implemented within the FE, the PDL, who was the PO, implemented the Refinements and managed the Product Backlog, whereas the NIT did not contribute to this theoretical-intended cross-team event. The lack of ability to provide a single Nexus Sprint Goal led to many Nexus-specific components being removed and ultimately the use of individual Scrum teams implemented Scrum, along with the NIT using the Nexus Daily Scrum to collaborate on a scaled level. Different systems were built by the individual Scrum teams, with each system or team contributing to the banking build product, but with limited integration where all the teams did not contribute towards a single Increment but rather individual team increments with the potential on inter-team work overlapping. The unique integration pattern at the FE should have

been considered and potentially have affected the way Nexus was viewed and implemented in the enterprise.

The FE could have followed a different approach in implementing Nexus, e.g. attempting some of the Nexus events more than once, since Scrum is about iterations and adaptation. The Nexus Sprint Review, as discussed in Chapter 7 could have been retried in smaller reviews where integration existed between teams, which would lead to better engagement and more focused meetings also raised by Paasivaara et al. (2012). Nexus Sprint Planning and Refinement should highlight and expose inter-team dependencies, but the FE's implementation did not provide the necessary context to demonstrate these events within the practical environment. The Nexus Sprint Retrospective also could not be reviewed in this study, as it was not really attempted.

The need to adapt Nexus is a natural and needed aspect, as Nexus is a framework that is meant to be adapted and moulded to fit the organisation, as stated in literature raised in Chapter 2 and confirmed by SM3: "*[B]ut from what I can see Nexus is a framework, it's meant to be adjusted; I think that sort-of encapsulates everything. Nexus is a framework, it's like Scrum, Scrum encourages you to stick to the rules initially and then deviate and adjust and bend the rules so to speak to a point where it works for your organisation*." "*Some things didn't work as well as we thought it would, but that was a learning curve; we inspected and adapted. So, Scrum and Nexus aren't solution-providing frameworks, they are impediment-exposing frameworks. They shouldn't be looked at as a silver bullet that is going to solve your problems, they should look at it as a thing that will shine a light on every single issue that we have so that we can address it head-on and improve*," reiterated by SM2. SM1 comments on the extent of adaptation that should be allowed in deviating from the theoretical framework: "*We've deviated to a point where we are running in silos; all of our teams are either doing some version of, either Scrum or Kanban, getting somewhere to something and hoping that's enough.* SM1 confirmed that the FE almost started from using a non-scaled version, to a scaled version, and then back to a non-scaled version: "*Yes. But now all that we have got is visibility during a Daily Scrum that says where there is a block or dependency.*"

Nexus, or any solution, is highly dependent on the enterprise setup, structure, and mindset. The FE did not have the support of management to adapt Nexus more, as it was viewed as a waste of time, and there was a strained relationship between the PO and the SMs. Management buy-in is a known success factor for agile frameworks to thrive in a practical environment. The lack of experience by those implementing the framework and thus a deficiency in coaching and overall implementation, are also contributing factors. External coaches or those with more expertise could have been hired to assist with the implementation of Nexus, as supported by literature to increase the success of an agile framework.

Other factors, such as standardising channels of communication and not implementing new technology while trying to simultaneously implement a scaled agile solution, were factors that negatively affected the Nexus implementation in the FE, but provide practical insights for practitioners when they implement a scaled agile solution in their own enterprise environment.

Overall, the elements from Nexus that were retained, such as the NIT, Nexus Daily Scrum, and Nexus Sprint Backlog (with only inter-team dependencies), did elucidate inter-team dependencies and assisted the enterprise in their main problem regarding inter-team communication and collaboration. As the PDL said, "*for us, in a Fintech business, and what we need to do, it was not necessarily a full Nexus approach, and I think we stole some elements of it and used it as a team. I think what actually did work very well, is that we used that as a platform to understand dependencies and inter-dependencies.*" Showing that this study does contribute to the literature gap regarding empirical evidence when implementing Nexus, this study provided insight about a real-world case study that attempted practical adaptations of the Nexus framework, focusing on addressing issues about inter-team dependencies, communication, and collaboration.

8.3 Recommendations for future research topics

The study identified some theoretical gaps regarding the Nexus framework that would be beneficial for future research, such as an implementation of Nexus where a single Nexus Sprint Goal could be defined, with the potential of implementing the Nexus Sprint Planning more effectively, aligned to the use of the theoretical Refinement process. A case study is suggested where more effort was given in trying to make the 'vanilla' Nexus components work and then iterating and adapting them.

Other non-specific Nexus topics that could lead to future research topics are: (1) grouping teams for globally-distributed teams working together on an agile or large-scale agile build, and (2) identifying the ideal number of members in an agile development team. The trade-off of grouping teams based on their location versus grouping them based on their skillset to create a globally distributed cross-functional team, and the factors and constraints to consider when creating such an agile development team should be investigated.

References

Agile, D. (n.d.). Introduction to Disciplined Agile Delivery (DAD). Retrieved from http://disciplinedagiledelivery.com/introduction-to-dad/.

Agile, S. (2018). *SAFe® 4.6 Introduction Overview of the Scaled Agile Framework® for Lean Enterprises*. Retrieved from https://www.scaledagile.com/resources/safe-whitepaper/.

Bass, J. M. (2016). Artefacts and agile method tailoring in large-scale offshore software development programmes. *Information and Software Technology, 75*, 1-16. doi:10.1016/j.infsof.2016.03.001.

Bresky, N. (2007). Book Review: Root Cause Analysis: Simplified Tools and Techniques. *Technometrics, 49*(3), 364.

Byrne, M. (2001). Interviewing as a data collection method. *AORN journal, 74*(2), 233-235.

Carmines, E., & Zeller, R. (2011). Reliability and Validity Assessment. In (pp. 17-27). doi:10.4135/9781412985642.

Cohn, M. (2010). *Succeeding with Agile: Software Development Using Scrum*. United States of America: Person Education, Inc.

Coleman, J. (2018). How do Nexus and LeSS Differ? Retrieved from https://www.scrum.org/resources/blog/how-do-nexus-and-less-differ.

CollabNet, & VersionOne. (2018). *The 12th State of Agile Report*. Retrieved from https://agilebb.nl/wp-content/uploads/2018/04/versionone-12th-annual-state-of-agile-report.pdf.

Company, T. L. (2019). Introduction to LeSS. Retrieved from https://less.works/less/framework/introduction.html.

Conboy, K., & Carroll, N. (2019). Implementing Large-Scale Agile Frameworks: Challenges and Recommendations. *IEEE Software, 36*(2), 44-50.

Connelly, L. M. (2012). Root cause analysis. *Medsurg nursing : official journal of the Academy of Medical-Surgical Nurses, 21*(5), 316, 313.

Cruz, R. F., & Berrol, C. F. (2019). *Dance/movement Therapists in Action : A Working Guide to Research Options* (Vol. Third edition). Springfield, Illinois, U.S.A.: Charles C Thomas Publisher.

Denning, S. (2015). Agile: it's time to put it to use to manage business complexity. *Strategy & Leadership, 43*(5), 10-17. doi:10.1108/SL-07-2015-0057.

Denning, S. (2017). The age of Agile. *Strategy & Leadership, 45*(1), 3-10. doi:10.1108/SL-12-2016-0086.

Digital.ai. (2021). *15th State of Agile Report*. Retrieved from https://digital.ai/resource-center/analyst-reports/state-of-agile-report.

Dikert, K., Paasivaara, M., & Lassenius, C. (2016). Challenges and success factors for large-scale agile transformations: A systematic literature review. *The Journal of Systems & Software, 119*, 87-108. doi:10.1016/j.jss.2016.06.013.

Dolman, R., & Spearman, S. (2017). ASK: AGILE SCALING KNOWLEDGE - THE MATRIX. Retrieved from http://www.agilescaling.org/ask-matrix.html.

Ebert, C., & Paasivaara, M. (2017). Scaling Agile. *IEEE Software, 34*(6). doi:10.1109/MS.2017.4121226.

Edison, H., Wang, X., & Conboy, K. (2021). Comparing methods for large-scale agile software development: A systematic literature review. *IEEE Transactions on Software Engineering*. doi:10.1109/TSE.2021.3069039.

Fourie, L., & De Vries, M. (2017). Exploring enhancements to the agile approach for mid-sized enterprises in the services sector. *South African Journal of Industrial Engineering, 28*(3), 29-39. doi:10.7166/28-3-1837.

Francino, Y. (n.d). Large-scale agile frameworks compared: SAFe vs DAD. *Big agile.* Retrieved from https://techbeacon.com/app-dev-testing/large-scale-agile-frameworks-compared-safe-vs-dad

Gill, A., & Henderson-Sellers, B. (2006). *Measuring agility and adaptibility of agile methods: A 4 dimensional analytical tool.* Paper presented at the The IADIS international conference on applied computing 2006.

Gravetter, F. J., & Forzano, L.-A. B. (2012). *Research Methods for the Behavioral Sciences* (4th ed.). Canada: Wadsworth Cengage Learning.

Guest, G., MacQueen, K. M., & Namey, E. E. (2012). *Applied thematic analysis*. Thousand Oaks, California: Sage.

Hanssen, G. K. (2011). Agile software product line engineering: enabling factors. *Software: Practice and Experience, 41*(8), 883-897. doi:10.1002/spe.1064.

Harrell, M. C., & Bradley, M. A. (2009). *Data collection methods. Semi-structured interviews and focus groups*. Retrieved from Rand National Defense Research Inst santa monica ca.

Hofstee, E. (2006). *Constructing a good dissertation: A practical guide to finishing a Master's, MBA or PhD on schedule*. Sandton, South Africa: EPE.

Holland, S. (2021). Type I and Type II Errors. Retrieved from http://strata.uga.edu/8370/lecturenotes/errors.html.

Hoogervorst, J. A. P. (2018). *PRACTICING ENTERPRISE GOVERNANCE AND ENTERPRISE ENGINEERING : applying the employee-centric... theory of organization*. [S.l.] :: SPRINGER INTERNATIONAL PU.

Kalenda, M. (2017). *Scaling Agile Software Development in Large Organizations.* Masaryk University, Retrieved from https://is.muni.cz/th/410499/fi_m/masters_thesis.pdf.

Kalenda, M., Hyna, P., & Rossi, B. (2018). Scaling agile in large organizations: Practices, challenges, and success factors. *Journal of Software: Evolution and Process, 30*(10). doi:10.1002/smr.1954.

Kawulich, B. B. (2005). *Participant observation as a data collection method.* Paper presented at the Forum Qualitative Sozialforschung/Forum: Qualitative Social Research.

Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2008). Systematic literature reviews in software engineering - A systematic

literature review. *Information and Software Technology, 51*(1), 7-15. doi:10.1016/j.infsof.2008.09.009.

Kurt Bittner, P. K., Dave West. (2018). *The Nexus Framework for Scaling Scrum*. United States of America: Pearson Education.

Larman, C., & Vodde, B. (2010). *Practices for Scaling Lean & Agile DevelopmentLarge, Multisite, and OffshoremProduct Development with Large-Scale Scrum*. Retrieved from http://ptgmedia.pearsoncmg.com/images/9780321636409/samplepages/0321636406.pdf.

Middleton, F. (2022). The 4 Types of Validity | Explained with Easy Examples. Retrieved from https://www.scribbr.com/methodology/types-of-validity/.

Moe, N. B., Šmite, D., Paasivaara, M., & Lassenius, C. (2021). Finding the sweet spot for organizational control and team autonomy in large-scale agile software development. *Empirical Software Engineering : An International Journal, 26*(5). doi:10.1007/s10664-021-09967-3

Okoli, C. (2015). A guide to conducting a standalone systematic literature review. *Communications of the Association for Information Systems, 37*(1), 879-910. Retrieved from https://aisel.aisnet.org/cais/vol37/iss1/43/.

Okoli, C., & Schabram, K. (2010). A Guide to Conducting a Systematic Literature Review of Information Systems Research. Retrieved from http://sprouts.aisnet.org/10-26

Paasivaara, M. (2017, 22-23 May 2017). *Adopting SAFe to Scale Agile in a Globally Distributed Organization.* Paper presented at the 2017 IEEE 12th International Conference on Global Software Engineering (ICGSE).

Paasivaara, M., & Lassenius, C. (2014, 28 July-1 Aug. 2014). *Deepening Our Understanding of Communities of Practice in Large-Scale Agile Development.* Paper presented at the 2014 Agile Conference.

Paasivaara, M., Lassenius, C., & Heikkilä, V. T. (2012, 20-21 Sept. 2012). *Inter-team coordination in large-scale globally distributed scrum: Do Scrum-of-Scrums really work?* Paper presented at the Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement.

Paasivaara, M., Lassenius, C., Heikkilä, V. T., Dikert, K., & Engblom, C. (2013, 26-29 Aug. 2013). *Integrating Global Sites into the Lean and Agile Transformation at Ericsson.* Paper presented at the 2013 IEEE 8th International Conference on Global Software Engineering.

Prikladnicki, R., Lassenius, C., Tian, E., & Carver, J. C. (2016). Trends in Agile: Perspectives from the Practitioners. *IEEE Software, 33*(6). doi:10.1109/MS.2016.152

QASymphony. (2018). The Pros and Cons of the Scaled Agile Framework (SAFe). *Agile.* Retrieved from https://www.qasymphony.com/blog/pros-cons-scaled-agile-framework-safe/.

Reifer, D. J., Maurer, F., & Erdogmus, H. (2003). Scaling agile methods. *IEEE Software, 20*(4). doi:10.1109/MS.2003.1207448

Schaffernicht, M. (2010). Causal loop diagrams between structure and behaviour: A critical analysis of the relationship between polarity, behaviour and events. *Systems Research and Behavioral Science, 27*(6), 653-666. doi:10.1002/sres.1018.

Schwaber, K. (2018). *Nexus™ GuideThe Definitive Guide to scaling Scrum with Nexus: The Rules of the Game* (pp. 11). Retrieved from https://scrumorg-website-prod.s3.amazonaws.com/drupal/2018-01/2018-Nexus-Guide-English_0.pdf?nexus-file=https%3A%2F%2Fscrumorg-website-prod.s3.amazonaws.com%2Fdrupal%2F2018-01%2F2018-Nexus-Guide-English_0.pdf.

Scrum.org. (2017). Security Software Product Company Uses Nexus ™ Framework. 3. Retrieved from https://scrumorg-website-prod.s3.amazonaws.com/drupal/2017-07/Security%20Product%20Company%20Nexus%20Case%20Study_0.pdf.

Scrum.org. (2018a). Cathay Pacific Airways Makes Nexus Their Official Scaling Framework For IT: Increases Product Delivery by 200%. 3. Retrieved from https://scrumorg-website-prod.s3.amazonaws.com/drupal/2018-09/Cathay-Pacific-Airways_Sept2018.pdf.

Scrum.org. (2018b). How Net Health Scales Scrum with the Nexus Framework. 3. Retrieved from https://scrumorg-website-prod.s3.amazonaws.com/drupal/2018-02/Case-Study_NetHealth_February2018_web.pdf.

Scrum.org. (2021a). *The Nexus™ GuideThe Definitive Guide to Scaling Scrum with Nexus* (pp. 10). Retrieved from https://scrumorg-website-prod.s3.amazonaws.com/drupal/2021-01/NexusGuide%202021_0.pdf?nexus-file=https%3A%2F%2Fscrumorg-website-prod.s3.amazonaws.com%2Fdrupal%2F2021-01%2FNexusGuide%25202021_0.pdf.

Scrum.org. (2021b). What is Scrum? Retrieved from https://www.scrum.org/resources/what-is-scrum.

Spector, J. M., Christensen, D. L., Sioutine, A. V., & McCormack, D. (2001). Models and simulations for learning in complex domains: using causal loop diagrams for assessment and evaluation. *Computers in Human Behavior, 17*(5), 517-545. doi:10.1016/S0747-5632(01)00025-5.

Uludağ, Ö., Philipp, P., Putta, A., Paasivaara, M., Lassenius, C., & Matthes, F. (2020). Revealing the state-of-the-art in large-scale agile development: A systematic mapping study. *arXiv preprint arXiv:2007.05578*. Retrieved from https://arxiv.org/abs/2007.05578.

Uludağ, Ö., Putta, A., Paasivaara, M., & Matthes, F. (2021, 2021). *Evolution of the Agile Scaling Frameworks.* Paper presented at the Agile Processes in Software Engineering and Extreme Programming, Cham.

van Wessel, R. M., Kroon, P., & de Vries, H. J. (2021). Scaling Agile Company-Wide: The Organizational Challenge of Combining Agile-Scaling Frameworks and Enterprise Architecture in Service Companies. *IEEE Transactions on Engineering Management*.

Verma, R. (2017). I'm Not Calling Your Baby Ugly - Two Ways and 25 Dimensions to Compare Agile Scaling Frameworks. Retrieved from https://www.scrum.org/resources/blog/im-not-calling-your-baby-ugly-two-ways-and-25-dimensions-compare-agile-scaling.

Vitral, R. W. F., Campos, M. J. S., & Fraga, M. R. (2013). The null hypothesis. *American Journal of Orthodontics and Dentofacial Orthopedics, 144*(4), 498-499. doi:https://doi.org/10.1016/j.ajodo.2013.08.010.

Williams, L., & Cockburn, A. (2003). Agile software development: it's about feedback and change. *Computer, 36*(6). doi:10.1109/MC.2003.1204373.

Yin, R. (2000). Rival explanations as an alternative to reforms as experiments. L. Bickman (Ed.) Validity and social experimentation: Donald Campbell's legacy. In: Thousand Oaks, CA: Sage.

Yin, R. (2018). *Case study research and applications: design and methods* (6 ed.). Los Angeles: SAGE Publications, Inc.

Ylikoski, P., & Zahle, J. (2019). Case study research in the social sciences. *Studies in History and Philosophy of Science Part A, 78*, 1-4. doi:https://doi.org/10.1016/j.shpsa.2019.10.003.

Appendix A: Interview transcriptions

This section provides detailed interview transcripts per interviewee, also indicating the two different data-gathering cycles, i.e. before implementing Nexus (Pre-Nexus) and after implementing Nexus (Post-Nexus).

a. Transcript of interview with Product Development Lead

Please find the interview transcripts below. The blue text represents the questions asked, the black text represents the answers to the questions, and any green text shows where the interviewer has asked something within a pre-defined question. Any text which is in square brackets masks the information given in the interview to hide any enterprise specific naming or employees names.

i. Pre-Nexus

What is idea of a 'euphoric' work set up in terms of UI, UX, BA, Team Leads and all of that; and how does it differ within the Hello Group?

When we started off it was all separate, BAs would be on their own, Devs would be on their own, UX would be on their own, Marketing would be on their own; but as we evolved everybody on the team did what they needed to pull together, so it became more like a team rather than individual skill set. In the team you have some people with strong skills but nothing stopped them from swapping from reading actual scripts and writing the scripts versus actual testing and doing the testing, versus making the BPMN model; and also then giving input on how the UX screens should look and work.

What system was being used?

We were using Scrum, but in isolated manners.

What was and wasn't working with that system, and why?

What was working with that current system was, it made sense for what they were doing at that point in time. Small, minor, changes on systems, focused on them and executed them, not a lot of involvement with other people; versus what we do now, where its massive systems big overall, and complete redesign which needed something like what we're doing now. *What wasn't working with that system?* I think there was continuous back and forth between business designs and actual dev, although that still exists, I feel its lesser than in comparison to the previous.

What were the causes of those issues?

I think it's the lack of Tech understanding. *Okay, from whose side?* On the business side, the people involved in actually designing the processes and putting screens together, had almost close to none tech experience. Whereas now, where our BAs and UX guys are fully involved in tech, so from day one of planning everybody understands what needs to happen.

Was there a need from the Stakeholders for a solution to these issues?

We had many teams, all working in isolation, there were dependencies that they had amongst each other; but those dependencies weren't clearly communicated and understood, as a result impacted their own deliverables. So, introducing the Nexus type-of approach, helped up get integration understood and dependencies understood between teams.

What caused Hello Group to consider the scaled-scrum solution.

The size of the project. *That's it?* And the many teams involved, there were 6 systems we needed to build.

Do you know if there were any other methods of scaled-scrum/agile looked at?

No

ii. Post-Nexus

Why was Nexus chosen, do you know what factors were considered?

Well when Nexus was brought to me as the approach, I basically explained the scenario of many teams to deal with and as Product Owner we need to have some form of integration, to understand the single-view of where everything is and what's happening. The idea of Nexus was then proposed, meant to be the group of people who keep all of these environments stuck together and also have a single view of where everybody's tickets are, across all teams.

How was this proposed to the company?

We had a workshop and went through all of the challenges and then the Scrum Master explained this type of approach and we decided to try it.

How was the adoption to this new system taken?

I think the adoption was fine, people understood what needed to happen and they did what they needed to within their individual teams and within the scrum.

What was and wasn't working with that system, and why?

I think that that the whole Nexus idea of a team doesn't work, Nexus is not a team. *Explain that.* So a team meaning, a team of people with different skills on the team and then setting team objectives on what our objective as a team is going to be. That does not work, because the

way that we have implemented Nexus is, we called it Nexus; but it was basically all the leads from the different teams, so each team has their Sprint objectives to meet, met on a daily basis to share any integration problems. Discussing integration problems and challenges between my system talking to your system, it's not a team. And secondly, we can't have a team objective. *When you say 'team objective', you mean like a Sprint goal for all 6 teams?* Yes, it's impossible. So what often Nexus talks about is having a Sprint objective for Nexus. Now it was really difficult to have a Sprint objective for Nexus. *We tried in the beginning.* We did try, and I could never have an objective, so I always gave the Scrum Master 5 or 6 objectives, an objective per team. So that doesn't do Nexus, therefore for me Nexus is not a team; maybe in other organisations the way they have implemented it, it would work as a team. For us, in a Fintech business, and what we need to do, it was not necessarily a full Nexus approach, and I think we stole some elements of it and used it as a team. I think what actually did work very well, is that we used that as a platform to understand dependencies and inter-dependencies. *What has worked in Nexus then?* What has worked was getting the leads, across the other teams to communicate clearly each day and fully understand where the blockers are amongst teams, and then ensure that the team responsible for that specific item has that on their priority list. Because often what happens is that I don't know it is a priority to you, so I'll just, within my team, make this a low priority. But if I know how important this is, because you explain to me in the daily Nexus meeting how important it is for you; then it will be easier for us to resolve it and fix it.

What factors do you think, if changed, would have helped these issues?

I think the issues we had weren't really a problem with the framework, I think it was more the understanding and the implementation that is actually realised in our organisation. The Scrum Masters involved, basically wanted to follow theoretical approaches and not then go back and look at the practical elements. I think from a dev perspective we were absolutely fine; from a build perspective we were absolutely fine. I think the Scrum Masters spent too much of time analysing the theory and implement theoretical components of what Nexus spoke about; in comparison to saying, "Here's guidance, which they understand, "and let's make it work for our organisation practically."

What were the main learning/lessons from this Nexus Build?

My main lessons is, ensuring that across the different systems, our specs are fully locked down. *Business specs or tech specs?* Business Specs, are fully locked down, the timeframes which are set to actually build this is fully understood by all parties, adequate time given for regressions testing; across the board. Also, in the initial days, Nexus wasn't what it is now. Nexus was every team member coming to give an update on their team. We changed that, there's only dependencies and we've got a Trello board which then didn't take us to each team, but what's

on the dependencies; after tracking that, that worked. So what would we do differently the next time, is that we'll start off with the proper board that says these are the Nexus dependencies, across the entire teams. *So, would you say it's a lot high focus on dependencies and integration?* Yes

What was changed or adapted from the original Nexus implementation to the now?

So what we changed was we were initially focusing of the details within each team, which was a waste of time because it was a repeat of what actually happened in their own Stand-Ups, which we didn't really need. So, when we changed the approach to just the dependencies, and tracked them on a Trello board, and moved away from Jira; it worked perfectly.

If you knew everything you know now, would you do Nexus again? If so, what would you change, from implementation to general running? If not, what would you consider implementing instead?

I don't know what I would consider implementing, but I definitely wouldn't go with the theoretical approach of Nexus, that doesn't work. What works is, on a project like ours, the scale at which we work at, you do need that daily collaboration between all leads on understanding dependencies; because they will always exist between different teams, it also becomes the touch-point to share important information, so that we'll do. So I would have also pushed back on say, a lot of hours were wasted on Nexus Planning. So you'll have a two week sprint; you'll have one full day of Nexus Planning, you'll have another two to three days of individual team planning, questions coming up, very little time for devs to dev, and then all of a sudden you'll have demo and a retro. So Nexus planning doesn't really need to happen, individual team planning needs to happen; dependencies that come out of your team planning needs to be addressed at the Nexus level and we monitor the rest of those dependencies. *When you say the team planning needs to happen, surely some information needs to come from that Nexus, that hierarchical, that top management team?* No, so it's not a management team, Nexus is not a team and it's not management, Nexus is, in my view, maybe it was the wrong word, and we shouldn't have even used Nexus, because it's meant to be a collaboration between all leads, that's what we needed. Maybe we went down this whole Nexus path, we should never have been on, but it so happened it was called Nexus and we continued that name and we tailored it. But all we needed in our organisation is a time-slot, or better collaboration, between all leads and the dependencies. How do you come about it, in the individual team, you would have said 'I want to do x, y, and z', not when the team sat down and said 'this is what we need to do on the system'. Let's take the CRM system for example, CRM said I need to do [the FE] transaction assist, then you said okay, how do we do this. *Okay so where would they get the requirement for [the FE] transaction assist, from management?* CRM as the system has

requirements, but the requirements within the CRM system, the functionality, has a dependency on another team; so that dependency is what should come back as a Nexus dependency, or whatever, a Leadership dependency or Integration dependency, or whatever one calls it. But you don't use Nexus to define what you're going to be doing across our environment, that's impossible. What we'll do is, as business, we'll say 'I want to build the bank', that's what we said we want to do, what do we need as business to build a bank? We need to so x, y, and z on our different systems, then we'd go to the devs on those systems, and say 'on your system this is what I need you to do'. The business would do their stuff, not the other way around, then now once business said 'okay for CRM this is what I need'. They'll come to the team building the CRM and say 'this is what I need you to do, let's start putting the backlog together ' and bringing in the different steps. When the team lead of CRM and his team are breaking down their stuff, they'll say 'What can I do myself, what do I need somebody else for', then say 'oh, this here, I need to go and get an API from the core transactional products; whose the owner of that, oh it's [team member]'; highlight that, come to Nexus, and say to the team dependency I need this.

In general, what is your personal opinion of scaled-Scrum/agile?

I don't have true idea of it because I don't think that's what we've implemented. We took the word scaled-Scrum and thought that's what we were doing. We followed Scrum methodologies within the teams and then tried to take this thing and scale it, which is not possible, it makes no sense in our organisational context.

Lastly, do you think Nexus was a success, with respect to this launch MVP Bank Build?

I don't know, it's a tough one Lauren, because I don't really think we really applied a lot of those Nexus principles; we applied them within the individual teams and that's where they should stay. We tried to apply them at scale level, and it failed miserably, it made no sense. When I say it failed miserably, for me it was just a waste of time; planning was a waste of time, retro is a waste of time, any discussion we had outside of just a catch up was a waste of time. So, a lot of time was wasted trying to take something and use it at a scale level. For the purposes of our project, I don't think that's what we needed. What we did need was a collaboration on each dependency, which is what we did, and we so happened to call it Nexus. Because what is Nexus? Nexus is the same thing you'd do with an individual team but at a scale level; it means you do the planning, you do the backlog, you did the execution, you did the retro; it doesn't make sense doing it at that level. *Can I just ask, why didn't it actually work at that level?* It didn't work at that level because we were building six different systems, with six different objectives and each sprint planning for each of those teams are different. *So basically, what you are saying is only parts of the Nexus idea was that collaborative Daily Stand-Up which we*

turned around to only do integrations, we don't do the "what have I done, what am I doing and what's going to happen" because that didn't work for us either? No, we didn't find the need for it.

b. Transcript of interview with Lead Technical Manager

i. Pre-Nexus

What is your version of a euphoric work environment setup in terms of team leads, devs, business, etc. and how does that differ to what we have in the Hello Group?

The ideal would be if everyone was co-located because a lot of the problems we had initially was because we had remote teams and it's hard to keep track of what everyone is working on every day. So, I only get a chance to talk to the remote teams once or twice a day and, in a project this size, it's not quite enough to keep track of everything. Either certain things that they need from me get missed because there's that time delay, or if I need something from them, I have to wait for an email or for them to be available; if they're here, I can just go to them.

What system was being used to try and build Bank before Nexus?

It was agile, there were a lot of individual team Stand-Ups and then we had one overall stand-up. It wasn't quite Nexus in the way we had defined it, it was just a stand-up with the leads. That meeting basically became a meeting where everyone just regurgitated what happened in the team stand-up; so, my team did this and today where going to do this. It wasn't really adding much value to each person. You had to pay attention to every single thing that was happening, then figure out what impacts you and what you need to be looking out for. In a lot of cases people would just come there, say what they need and then leave, no one really discussed anything. The follow-up also wasn't really there in the sense that if you came back the next day and said you were still working on the same thing no one would question it because it's separate teams.

These are two different questions, but you can answer them together if it makes it easier. What was working with that system? And what wasn't working with that system?

What was working was, within the teams they were doing okay, the inter-team dependency though was where things started to break down a little bit. It was hard to coordinate between teams, especially when deliverables were due, and a team was dependent on two or three other teams. They might be able to work directly with one team, because we have two teams in India and then a couple teams in South Africa; if you were here and you were dependent on everyone here it was relatively easy to get things going. If you were dependent on teams in India, then you were locked to their dev cycle; they had already decided what they were going to do in their sprint before, they had no idea that you were looking for anything from them. Now, it's

better in the sense that we, every day, can tell other teams what we need, and they can adjust their sprints accordingly. *Were the issues then just because of the global distribution or what would you say the issues were?* It was a bit of the global distribution and the teams had different Scrum Masters as well, so the Scrum Masters would get together and talk but they wouldn't be able to discuss the details of what the teams were doing, there would just be an overall "this is where we are at" type of thing. Since they aren't involved in the actual building, the development work itself, they don't have the understanding of all the possible dependencies that could exist.

Was there a need to sort these issues out for the Bank build and why?

Some of the issues that we had, we picked up when we were trying to integrate between the different teams. We realized pretty quickly that what one team was dependent on, another team would only get to in their next Sprint, or the other way around; the backend team would be done with something but they couldn't take it live and test it because the other teams weren't ready to use it. So, there a few disconnects like that and we realized that we needed to make that process a little bit better.

What caused Hello Group to look at a scaled Scrum solution?

[Scrum Master 2] can answer that. We tried it out and it seems to work so far. The thing with agile, Scrum really, is there is no fixed process that will work for everyone so we were just taking the standard; the standard in the sense that it's been tried by various companies and this is what people think works and we try it out and, if it works we keep using it, if it doesn't we adjust it or stop using it. Similar to the way we were doing the scrums beforehand, we had the scrums within the teams and then we had the scrum of scrums, which is the acceptable way to do it. If you had a team this distributed, this large, working on different aspects of it then…you see Nexus is still really a Scrum-of-Scrums, it's just not calling it that and, since we're not calling it Scrum meeting people approach it a bit differently.

I'm not sure if you can answer this, but do you know, other than Nexus, what other methods of scale Scrum were looked at?

No.

ii. Post-Nexus

Why was Nexus chosen? What factors were considered?

We didn't really consider any factors really. It was a suggestion by [Scum Master 2] and the other Scrum Masters, so we agreed to try it out and see. We didn't go and look at all the available

options, look at all the pros and cons, etc. We were given a possible way and we decided to try it.

How was this proposed to business?

Well, it was actually proposed to us during one of our, at that time, Scrum-of-Scrums meetings. We had everyone there asked them if this was working for them, was this meeting providing them with any value? If not, what could we change about it and what would they like to get out of it; the outcome of that was they didn't really need to know the details of what each team was doing, they just needed to know what they needed to do for other teams. *Out of interest, who was involved in the scrum-of-scrums meetings?* It would have been all of the Team Leads. *And the Product Owners?* Yes, they would be there too. They would also attend some of the team Stand-Ups and I would too, but then I only started attending the Scrum-of-Scrums one and the web service one because they were giving the same updates, so it made things simpler.

How was the adoption of the teams to Nexus?

It was quite good. The first couple of meetings were a little bit of a repeat so we had to remind them to stick to what the dependencies were. We then started the…we were still doing it on Jira so that we would see the epics, we would go through those; for each epic, what are the dependencies? Then we switched to the Trello board, a Kanban type style was better for that because it's not really stuff for an epic necessarily, it's a little task that needs to be done or a bit of work that needs to be done and doesn't need to involve the whole team necessarily. It can be done by one person, for example. So, we just have that board and we discuss that. So, at the moment, it is getting rather large in the sense that there are lots of things in progress, we could prune that a little bit as a lot of the things would move to blocked or we should had a "Do Later" column or something like that, but we will see what happens over the next few weeks how it goes. There is still going to be a lot of other things that we will be adding onto the bank. *Yes, this is just the MVP launch, so there is still lots to come.* Yes.

What has worked and what hasn't worked with Nexus?

What has worked is the collaboration between teams is better. What hasn't worked is the whole thing with the Scrum-of-Scrums and Nexus and all that, is that the aim was to speed up the pace of delivery; that sort hasn't happened as we wanted it to. It has somewhat in the sense that it's a bit smoother, but it still doesn't help the teams work any faster as the dependencies are still there, so it takes time to resolve them and things like that. I think it'll come down to, as the teams become more familiar with how they are building their systems and get more onto the operational side of the systems and the building; everything is in dev-mode at the moment so there is nothing really in production so there are no production bugs to sort out and that, but

then once that starts happening then the teams will be a bit more split. You will have one or two people in the team looking at operational issues, other people looking at dev, and those two parts of the teams will also be working together; if they pick up something in production, it means it will feed back into dev and so on.

What factors do you think, if changed, would have eased some of the issues we had when we first started Nexus to now?

Changed in Nexus or in the teams? *Well, any factors that contributed to the issues we had with Nexus. If they changed, would have helped some of those issues.* The one that stands out is the move to the Kanban board for all the issues that were happening with Nexus. I think that's the one that provided us with the most benefit. *Even things that didn't change, if they had changed, would've helped us. Even if they haven't been implemented yet.* I can't think of any right now actually. Other than what we spoke about earlier, that's about it.

What would you say are the main lessons and learnings taken out of this Nexus build?

I suppose in the fact that the dependency between teams have to be catered for in the planning as well. So, what we might have to do is, when the teams have their planning sessions, as part of their initial planning session they need to also determine what potential dependencies they would have for this sprint and then communicate this in the next Nexus session or actually… If the planning between the teams are staggered, they could have a representative in the other team's planning session as well so that they can motivate why their stuff is also important. It is possible to do it through the Scrum Masters if they are there as well, usually it would be the product owner; so, if there are multiple teams working on something then the product owner would attend those planning sessions and say what the dependencies are. With this many teams though, it's not always easy for one product owner to do all of that. So, we'll get better and that and see how we can make it work.

What was changed or adapted from the original implementation of Nexus to how we run it now?

Nothing too drastic changed. The big change was, when we started Nexus, people were treating it as if it was sort of an abbreviated Scrum-of-Scums, but then we changed that, we got the Trello board in, and now it's more of a discussion of what is happening. In a Stand-Up, generally everyone would have something to say, now in the Nexus it's just a matter of asking if you are dependent or blocked by anything, so a lot of the teams now just say no we are okay. When we started, every person felt like they were obligated to say something, because there has to be something happening kind of thing.

If you knew everything you knew now, would you do Nexus again. If so, what would you change. If not? What would you consider implementing instead?

I'd still do Nexus. I don't know if it is the best way of how to organize teams like this, but it is a way and it's working for us right now so we can keep doing it. If we pick up any really big issues with it and realize it's not working for us, then we will try something else. It's not really a fixed thing, we are not focusing any form of legit software development practice as such; the principles we stick to, but other than that it's up to us to do it however we want.

In general, what is your personal opinion on scaled Scrum or agile? Remember, Nexus is a framework that links to scale Scrum.

The thing is that, if you are getting to a point where you have to scale Scrum, then the project itself needs to be evaluated in the sense that the way that the project is being built basically becomes this massive project across many different teams. We should look whether the project can be built in modules, so each team, smaller team, is responsible for a module and then we have defined interfaces between them. The thing is that the way we are building the bank, since it is a departure from the existing technology that we have, we are sort of building the base for that thing in the future. So, now with every team having a web service, it basically means that you can have that modularity where if a team needs to implement a new feature, they just need to make sure that the web service that they have published still works and then they can add new features onto a new version of the web service, and then whoever else needs to implement that feature, can just use a new web service, whereas using the old web service should still be able to go on. But that will only happen in another six months or so when the project is more mature. *So, this was quite a big piece because this was the foundation to get us to that point?* Yes, but the thing is that, that point will always be a target to get to, right? Every new thing that we build, will have to be built in that way. The other problem that we have is we don't have enough people to have just one team, maybe four to five people, responsible for just one set of web services. There is always ten or twelve people who do six different things. The thing is, I don't know if we will ever get to that sort of scale. If you look at how Amazon does it, their model is that each team has their own service and they are responsible for everything for that service; the design, the development, the deployment, production support, everything. *So, they have basically taken out that integration factor?* No, the integration factor is still there, but there is no one overall team looking at how everything is deployed, each team does their own thing. There is a team that provides all the underlying hardware, the base on top of which anyone can deploy a virtual machine, but then the team themselves can decide themselves as to the architecture of their particular component, what technologies they use, what database they use, it doesn't matter as long as the interface they are providing, the web service, the API, etc. is standard. *When you say they don't, there is still obviously integration, but then who deals with the integration?* But now the integration, let's say you are building a product now, you know

what they product needs to do and can go and look at the repository of all the web services that are available across the company and pick and choose which ones you want to use. Each one of those has got the documentation and everything you need to help you understand how to use it and you just use it. If you pick up a bug in that system, then you just report it to that team, and they would have to go fix it. *Okay, so it's quite structured, in terms of all web services have to have the correct documentation, have to follow specific guidelines, etc.* Yes, so we are sort of getting there with the API gateway we have, all APIs available are there and you can go into it and see all the functions that are available. We haven't yet put up all the documentation for the APIs, so you can't click and see that a specific API does this because of this, and what you need to do to start integrating with it. So, that part is still missing but we will get there. It's a lot of the documentation work that we still need to do but that we'll do as the teams get time. It will probably be after MVP, a few months after that, when things are running stably, then we can start focusing on those things. We are not as large as Amazon, you'll be working on projects where you have never met the other guy before even though you've been working in the company for a couple of years; we are not at that size yet, so a lot of the cases, a lot of the integration that we'll be doing, we can just talk directly with the team involved. *The documentation is quite nice in terms of a knowledge sharing base…* Yes, but we have to see how that works because the culture here is still that you go talk to the person. So, even if you have extensive documentation on the service, the team can still come to you. *So, when it comes to handovers, when someone leaves the company, how does that work in terms of knowledge sharing?* That would be more important, but then that will also be handover within the team, so it's only if an entire team leaves will we have a problem. In most cases, if someone leaves, the knowledge is still within their team.

Do you think Nexus was a success with respect to the launch of the MVP?

Yes, I think so. It's not absolutely perfect in the sense that…I mean the idea with Nexus is that everyone knows everything that's going on and there is transparency to what's happening in the teams and stuff like that; I don't think that's quite there yet but in terms of the transparency within the team, if you go into each team's Jira board you can see what they are doing, sort of, because the tickets that they put on usually only mean something to the team or just to the developer involved. It would be a one-line message on the ticket and then nothing in the description, so if someone else wants to take over, they won't know what the ticket means. So, a lot of tickets are still placeholders that they can just look at and know what they are doing but to get detail, you have to ask the developer. That's also a result of the size of the teams as well, if the teams are larger and there were a lot more people looking at the tickets, then there would be more incentive to put more detail into the ticket. Alternatively, if there are multiple people

working on a ticket, then they usually put more detail into that ticket such as code, or instructions on what to do. So, a lot of the tickets where I see a developer create a ticket and assign it to someone else, then there would be something in the ticket, but if they create it for themselves, then generally it's just a one-liner.

c. Transcript of interview with Senior Business Analyst

i. Pre-Nexus

At the beginning of this project, what system was being used?

We had Scrum within our small groups, that's about it. It didn't always work out though.

What was and wasn't working with that system?

Obviously the catching up on what everyone was doing, and I feel like the focus sometimes fizzles a little bit. For me, I like Scrum when I know where everybody else is and when I can get my questions answered and knowing what everyone is working on, because obviously communication is better. But I feel like Stand-Ups are not always used the way they are intended to be used, and I think a lot of things get lost.

For those issues that you mentioned, what was the cause of those issues?

I think maybe everybody has a different idea of what it is you should achieve in that place. The shortfalls for me were that we didn't have one vision of how to complete something, everybody had their own idea of what needed to be done. That is not something you can fix with a Stand-Up or Scrum, I don't think that is what we are supposed to do, that is what the boards are for, that is what when we all kind-of need to collaborate on one; but I also feel that is a 'people problem', as in it is everybody's approach to it and it is everybody's idea of how they want to solve things, it is not necessarily the system that is not working it is the way that it is used.

What was considered in the business realm, why did we feel the need to scale Scrum for Bank? What were the factor considered?

The thing is that you have to develop your Scrum, when you want to break it into smaller groups, you have to be very smart about it, because there needs to be an overlap in knowledge between the groups but also the communication. It shouldn't be a big group, or too small, all the relevant parties should be present. I also believe that because we were doing so many Stand-Ups and meetings, and if you do it right and you do the Retro; all these things take up so much of your time that you don't actually get to your work, and I think that was a big issue. People like myself, [Product Development Lead], and [Lead Technical Manager] need to be in included in all the teams, that's what your whole morning is like. It's better but we're just here for your information and that took up like your whole morning because you had to be in like 4 different

Stand-Ups in the morning, I would have liked it to be kind of consolidated, the point, and I think that is why they broke it up because a lot of people didn't get any use out of the framework as it was. Too many questions from these things of people who don't understand the context, who weren't involved; feel like they need to know, and always ask questions that are maybe not relevant and are difficult to explain, and it makes the scope a lot bigger. But if you had your team which are dedicated to this project, everybody's onboard and knows what's going on; and now someone in this team mentions something that someone in our team didn't know and it evolves into a whole discussion.

ii. Post-Nexus

Seeing that you were here from the beginning of this Bank project, how was the adoption to this new system, Nexus, taken?

It's quite a difficult to answer because I haven't been included since Nexus started, I only joined once it had been going on for a while. I think everyone had found their feet and it was well structured. Like I said before, the previous, bigger, version was a bit messy. So by the time I integrated into the group, we had already tried everything and I think people had the right mindset and I think this time it was made up of people in the right groups, so I think it wasn't too bad; although I can't really account for the start of it. *Okay, because obviously you're a BA so you would be designing systems and all of that, why weren't you included in Nexus from the beginning?* I think it was more because I wasn't necessarily a decision maker for some of these things, I know [team member] was involved and did the Stand-Ups for Onboarding, so when I took over they were like 'Okay you also have to be in Nexus'. I'm not really sure, I think I should have been, they didn't really understand or know the BA's role would be in all of this and if it would be necessary or if it would complicate things, make the scope bigger and the questions bigger. I was only told what was necessary, to the point where my work was integrated with whatever the devs were doing. I designed something, they started building it, and only when you get to that point do you actually start communicating, which is the wrong way to do it but that's how it was.

Looking at Nexus, and the Bank project, what has worked and why?

Only the most important people are included in the loop, the decision-makers, the people who have a say or the people who are in the middle of everything. I think that works because like I said, that makes the scope a bit smaller, our meetings are kind-of on time and to the point, everybody knows what's happening. I like the organisation of it, it feels like we do sometimes go off-point, but it does feel like a bit more structure and I think the communication is a lot better, although it could definitely improve a lot. It has helped with that feeling of being

organized, previously I feel like people had kind-of been running around like headless chickens, and this helps everybody build to towards that common goal.

What hasn't worked with Nexus and why?

I think the communication, although it has improved, is still not good. There is still lapse in communication regarding a lot of things, for example the dates, when are we going live? There are four different dates, nobody knows what the real dates are. There are still things being built somewhere, that people are not aware of. For example, there is an improvement, a functionality to be built, after MVP, but you speak about this now and other people assume that this is important. I think people communicating, or relaying a message, doesn't always get communicated correctly to the person getting the instructions. I found that a lot with myself. If I ask someone how is something working, they'll answer me and that is how I'll implement it; then when I show it to them, they are like 'we are not going to do that now' or 'maybe that's not the best way to build it', they always think that if you tell me this then that is what I'm building now. I feel like people assume a lot and think that's in Nexus, I feel like everybody still does a lot more of their own thing to get to their Point A or Point B, and it's understandable when you're under this much pressure that you don't collaborate as you should, maybe go into your own little tunnel vision just to get things out.

So, the issues that you raised now, what factors do you think if they were changed would have helped these issues?

It's a difficult question to answer because I would say to break it down even more, and maybe the integration between the smaller teams to Nexus should be better because it's a massive project and you can't have an overview of everything that is happening. So Nexus is kind-of like the overview and it seeps down into smaller teams, and I don't know how that could be improved. Because I think, the way that we're doing it is fine, the scope of this project is just so crazy that having your fingers on everything all the time is close to impossible.

What are the main lessons, or biggest learnings, you can take out of this build?

The one I've already mentioned is just assuming other people know what you are talking about is a big thing for me, I've found that a lot. I also feel like context is important, I've also asked questions before where people gave answers and I go back to implement; the answer is correct, but they are not looking at it from my point of view and that is why it won't work. So I feel a lot of time is wasted when you ask someone the question without explaining or sitting down with them and taking the time to do something; a lot of those cases I had to redo things that were already done. What I have learnt with the Compliance group was that you have to get your team organized, you have to give them this view of what we are building towards; because I

felt like so many weeks were wasted where everybody was so confused just trying to find their feet, not really know what it is we are building towards. Anybody buying into any idea, must be 100%, else it won't work; so it doesn't help you come up with an idea like Nexus when half of the team is kind-of in kind-of not, 'cause you have to have dedication, you have to have people that attend the meetings that follow up on what they are supposed to be doing and do what they are supposed to be doing. I've also learnt that timelines don't mean anything, timelines are just the roughest estimation you can go with.

What was changed or adapted from the original implementation of Nexus to the way it was run at the end of the project?

The only thing I would say, and it may be a weird answer, but the urgency of it. I think people didn't take it seriously at the moment because we had time left, and then people realized that if we don't get our work in order this is not going to happen; and I think that wasn't a strategic, maybe it was maybe there was strategy, I can't say there wasn't; but people took it more seriously and they realized this was on the only tool that would get us to pull together. People started taking notes, making the boards, keeping the board up to date.

Having your view of Nexus in this build, would you consider doing it again? Would you consider doing something else?

No, I think it's the better way to go. I do think that there are some improvements to make around how some things are structured and done. *Like what, can you give me examples?* Maybe the tools of collaboration, I still feel like things get a bit lost in all the channels we're using. It comes back to that question I couldn't answer about breaking it down into smaller parts and feeling like just this project is just too big to keep your tabs on. The communication should be better, and you should have a whole view of this thing but I do not know how you would achieve something like that.

Do you think Nexus was a success? When it comes to this build.

I do think so, yes it was, definitely, and I'm basing it on how far we previously got with the teams before, without it. Though I still think there is room for improvement, I still think it has helped.

d. Transcript of interview with Scrum Master 1

i. Pre-Nexus

In your opinion what is the main difference between Scrum and agile?

There's no difference. *So, they are the same thing?* No, sorry. Agile is a paradigm, which we implement as part of a move towards becoming better in a certain way. It's a mindset, it's a

thinking, it's a way of being leaner and cleaner. Scrum is a framework that we use to get us to agility. If you think that agility is a state of being, Scrum is the vehicle, the car, that we are going to get into and drive to get to the state of being, which is agility, which is agile. So, that's why I say, there's no difference in the sense that in implanting Scrum you should get to agility. But you can't just do agile; you can try to just do agile thinking, but it's always better to use a vehicle, and that vehicle we're using is Scrum. There are about 27 others, but we're just using Scrum.

Before you were considering scaling Scrum, what was the system that was being used?

We just used Scrum in parts, if I can put it that way. We were scaled, we were trying to instill and implement Scrum within teams, at ground-roots level because we wanted to try and implement from the ground up. So, if we got it into the teams, got the teams functioning in Scrum, so the reason (which will follow later) we picked Nexus was because the teams knew Scrum, so it made it easier to scale it. So, before that, we didn't have anything scaled; we didn't need anything scaled.

What was working with that current Scrum system?

I think, in pockets, we found that some of, most of, the teams had really good cadence; proved to deliver properly and well within Sprints; team morale was up; people were working well together; people were understanding strengths and weaknesses of each other; so the teams actually were working in a very happy way. You know teams go through, what we call, norming, forming, storming, and performing teams, okay? So, we got the teams when they were thinking they had formed, but what we did was we reset and we reformed, re-stormed and then re-normed and performed. So, some of the teams, got to performing before Scrum, or before we changed. Some stayed in norming-forming stages.

What wasn't working with that system?

I think it depended on how Scrum was explained or coached within some teams. So, some individuals are not particularly pro-change and like to do things a certain way. So Scrum is very team-focused, Scrum is very iterative and "let's get better", where some individuals prefer to work on their own, they just want to be little islands and they don't want to be a part of something bigger. In those teams, it didn't work, because those individuals would usually derail the team, or derail Scrum, or they would just make it really difficult to implement. Also, it was nice because teams were going along nicely but we had no visibility of what everyone was doing, so this team would deliver something, but other teams didn't know what they were delivering. Retrospectively, I think we should have perhaps contemplated a "show and tell"; show what someone has built, and then look at it. But we didn't think of that at the time. So, I

wouldn't say it wasn't working; in most instances where we did try Scrum, it did work. We just had one or two teams that would have taken longer to get there. *Okay. We've spoken about the causes of the system not working, and that was the people. And then you said there wasn't much visibility between the teams.* I'm trying to think if there was anything else. I think it just depended on the type of coaching that you got. Oh, management buy-in! This was huge. So, there's a difference between management saying that they buy in and management actually buying in. So management would say they are all for this, and give to go-ahead to do it, but then this same management are also not easily coached, and not easily flexible in the way that they do things. So, for them, they will give the go-ahead for you to go do your agile, but they want something now. They would then sabotage a Sprint, or they would sabotage a delivery, or whatever the case may be. So, it was kind of difficult to understand that management says we should do agile but then think, because we're agile, we can just switch. But actually no, if you think about how we want to deliver certain things, we're *this* close; we're 5cm out from a whole ruler-length; we could just as well finish it and then do the next thing. But management would still stop progress to do something else. The disruption was still; the pivots, were still there. That was caused by, usually, by management changing their minds.

What is your version of a "euphoric" work setup? In terms of teams, leads, etc. and how does that differ from how the Hello Group is set up?

Ideally, what I would like, is to have everyone that we have identified as…Okay, so according to Scrum, self-organisation is key; allowing individuals to select where they want to work and giving them the opportunity to decide that they will work in one place for a period of time and then thereafter have the option of moving. So, ideally what I would like to see, is teams formed according to cross-functional skill. So, we need this feature for this product, and we don't design a team around it. We say we are going to build eight features, and we advertise these to all the developers, testers, BAs and everyone involved; they then pick where they want to be. We then make sure that, within that selection, that everybody that is in that team is either a back-end, front-end, designer, BA and so on. Ideally, I would have liked to see in every team, if it's possible (but in this company it won't be possible) a Scrum Master and Product Owner in each team. We are lucky, because we actually have a Product Owner for the product, so he is kind of available most of the time and I think be BAs will be proxies. So ideally, that's what I would like to see happen. I'd like to see self-organisation, I'd like to see the teams deciding for themselves on how they are going to do it.

Okay, and how does that differ from what the Hello Group has?

In Hello Group, we were approached by the management and Product Owner and were told how they think it should look and how they think the teams should function. We did have some input into what we thought the team structure was, and if we agreed or disagreed, but it kind of was just a matter of placing the best person for a particular section in that section and didn't provide an opportunity to cross teams or change their structure. They also based it on tech lead skill in current teams, they based which team was best suited to which area based on what they are currently doing. We did give the people an opportunity, just after launching, to move and we had two movers that said they don't want to be where they are, and they were moved. It wasn't ideal though, I would've liked everybody to choose their own teams, I would have preferred that.

What caused the company to decide that Scrum wasn't going to work for this project, we need to do a scale-Scrum approach?

We were only two Scrum Masters and we realized that the magnitude of this product, having to logistically facilitate all the teams. So, what we needed to do was we needed to get, somehow, to a position where we could all work on one product and all the teams have visibility of each other. Within the agile world, having visibility of each other, and understanding what everyone is doing on one product is defined as scaled. *Okay, and that's what you also mentioned earlier. The problem that we had with Scrum was that, people individually were working great, but we had silos. There wasn't that visibility.* Yes. So, what we decided we would suggest to them something called Nexus, but we needed to scale so that we could actually take that which they already knew, Scrum, and just put it on another level but make the visibility between the teams so much better. That's what scale does, it makes everything on a higher level visible to everyone. So, you can understand why you are doing the piece you are doing within the bigger picture instead of just focusing on the one little piece within your team. *So, you understand the integration as to where it fits in?* Yes, you have to. You also understand how you become dependent on somebody; if you have a clear pond of water, or a bowl of water, and you just stick your finger in that bowl and you remove it; that ripple effect; you actually understand that better when you know what's happening between the teams. We had to scale, we had to find a way to make what we were doing on a small scale, on a big scale.

What methods for scale-Scrum were looked at?

Honestly? One. *Only Nexus?* Only Nexus. For a few reasons. Firstly, neither my colleague nor I know SAFe. I wouldn't recommend implementing SAFe if you don't know it; number one. Number two, it's pretty tricky and you need a certification in it. Secondly, we don't know LeSS either; neither of us knew LeSS, so we discarded it. What we did do was we went and did a lot of reading about how to scale one product on a Scrum method and Nexus seemed to be the best

method. Because we both come from that environment where we know Scrum really well and the teams know Scrum really well; we figured that if we could just add a layer of Scrum on top of that, it would make the most sense to use Nexus as a model. Then, what we started doing was having a look at Nexus and, the more we looked at Nexus, the more we realized it is a good fit for us. We first came up with the idea of what we thought was going to work for us, we then read about it, and then we came back and confirmed that it will work for us. We didn't really look at anything else, we just decided Nexus was going to be a good fit. That's it.

Do you know of any examples where Scrum at scale has been implemented successfully and, if you could, give examples? If they are confidential then you obviously can't.

Successfully, tried, or claimed? *All of them.* Okay, Momentum claims to have successfully implemented SAFe, yet to be proven. Multichoice claims to have successfully implemented SAFe, I think their claim might be closer to success than Momentum. Most places that try SAFe specifically, fail miserably, because SAFe is a very complicated method to implement. I know of nobody who has tried LeSS. FNB uses, what they call, DAD; how effective it is, I'm not sure. So, I don't know of a company that has said that they have implemented a particular scaling method and it worked! The closest I have come is Momentum saying that they have and Multichoice. That's just in Johannesburg.

ii. Post-Nexus

How was this proposed to the company?

My colleague and I literally just spoke to the Product Owner and said this is how we think we should scale it, and he gave the approval.

How was the adoption of the new system?

I'm trying to think. We didn't get everyone all in a room. I remember going to our teams and explaining what we were proposing; at that point I was Scrum Master of two teams and my colleague was a Scrum Master on three teams. We went to our teams and explained our proposal and that we would like to form a Nexus integration team, which is the key of Nexus. We said we would like the teams to select who would represent the teams in this Nexus integration team. So, at that first meeting, we then explained to them what the function of Nexus is, the Nexus integration team and how we as a Nexus integration team filter everything back into our teams. So, that was kind of how we proposed it to them. But the proposal literally went to the Product Owner and he then, either took it to senior management, or gave us the go-ahead directly. We didn't put everyone in a room together, we just went to our teams and requested representatives and then put them at the Nexus level, and then at that level, tried to guide them. *So how was this change taken?* I think it was positive, I really think it was positive. We made tweaks as we

went along, but things like daily Stand-Up we tweaked a lot to where it is now, but they really value that; they value that "How am I dependent on someone else?"; they really valued looking and seeing the silos and understanding the dependencies and how everyone integrates and depend on each other. So, they valued that visibility and transparency. When it wasn't there, when we decided not to do it, or when the Product Owner was rushed, then it made things muddy, caused conflict, and miscommunication. I think that the team accepted it well, the project team as a whole. No one has come up to us and complained about the idea.

So, with Nexus, what has worked and why? For this project.

What worked really well was, when we got the cadence right, it worked well. So, in Scrum, it's important to get your daily, weekly, monthly cadences. So, once we got to the point of every day, we were doing our daily Nexus-level stand-up, then it worked properly. We did Sprint Planning at the beginning of Sprint, ending Sprints with Reviews and Retrospectives and then starting again; when you get that cadence going very well, then it works well. Not only on the Nexus level but in the teams. It works well because that's how Scrum is designed. If anything goes wrong, you go back to basics and ask what Scrum says we should do and do that. So, I felt it opened relationships; where certain tech leads knew of other tech leads, and never worked with them before; this narrowed that gap, so the Nexus-level integration members really started working with each other a lot more but at the same time the team individuals were working with each other a lot more. Specifically, teams that were higher-dependency teams. What didn't work well was the minute we deviated. *Okay, describe that.* The minute we dropped a planning session, for example, we broke cadence. The minute we deviated anything from the basics, then it didn't work. So, the minute we didn't plan, the water became murky again and the teams didn't know what the other teams were doing and where they were at. In that whole time-period we only did one Nexus-level Review; we should have been getting better every three weeks, and if we had done that, we would have been a lot better off. That's the core of Scrum; inspect and adapt. So, we were not doing that well. There is always a push and pull between the Scrum Master and Product Owner, most time of which the Scrum Master lost and when that happened, we didn't have Retrospectives. So, the minute we deviated from the Scrum-set framework, or the set Nexus framework, it would derail. It has now derailed to the point where there is nothing really visible at the moment, we have to reset. *When is this reset planned for?*After the MVP go-live.

What factors do you think that, if changed, would have helped some of these issues?

I would have loved to have seen a Product Owner-Scrum Master relationship. This relationship would have had the two standing as a united front because often times there was no unity. The Product Owner should have been able to understand the pressure, but still made time for the

retrospectives and planning. So those type of things that fell by the waist side were because of there wasn't a unity in leading the teams. There isn't really anyone to blame, it's more of the fact that the coaching was not done with the Product Owner; I wasn't the lead on the product, so I didn't have the opportunity to do that coaching. I think that a person needs to understand that if the Product Owner and Scrum Master work well together then everyone will fall in underneath. If there is a push-and-pull the whole time between Business and Tech then Tech will always lose.

What were the main learnings/lessons that were taken from this implementation?

Do not use new technologies. Ever. What bit us badly was, if we had the technologies we always had, that we knew and that we worked with; and we had started Nexus on top of it, we would have delivered far earlier. I'm sure of it. Also, we did a new introduction of scaling and a new introduction of new technologies, what took really long was understanding the technologies and then building the bank. So, my biggest learning is, if you are going to do one or the other, put the new technologies in place and then tell the leaders that it is going to take three months to get used of the technologies; make hard decisions fast. We didn't do this, we waited nine months to ask whether the technology was working for us, and when one didn't, we kept it in the mix because it was too late in the game to change anything. Now we are working around it, trying to make it fit. We are forcing a square peg into a round hole. So, don't do that, don't try to introduce new technologies and scale at the same time. Just do scale. The other really important thing that I learnt was, for the Nexus integration team, at that level there should be a dedicated Product Owner and a dedicated Scrum Master that just focus on that team. That way, they make sure that the cadence of that team is working well which filters into the cadence of the other teams. That's my two biggest learnings; there were several others. *What were they? You don't need to go into depth.* If you are doing your daily Stand-Up, find a format that makes blockers and dependencies visible fast. State what your blocker is and assign someone to help, quick. Find that first. In a daily Stand-Up it's really not about what you are doing and your progress, it's about "How am I blocked and who's blocking me?" and getting that person in the daily Stand-Up of the other team to commit to assisting you and getting it out of the way. *Are you talking at a team level or Nexus integration level?* Nexus integration level, because what would happen is one team would be blocked and they would stand there and state that they were blocked but didn't do much about it. It took us a very long time, we went running around for eight months and now, in the last two months, we have got this blocker-dependency board and that has helped us, that has given us the visibility. So that's what we should have done that earlier. Make that visible fast, get your Stand-Up to work in that fashion. Do your planning. On a Nexus integration level, do the planning; take your time. Get it on a board, user-map it, take

string and red pens and make the dependencies visible; do the planning. Otherwise, there is no visibility, your visibility gets even more murky. If you don't do the planning, then the teams don't know where they are at. When you get to the actual Nexus Sprint Planning, then you do your Refinement. Walk out of the room with a goal, an attainable goal for a sprint. We were not strict enough on implementing those basics. Inspection and adaption are the crux of Scrum. If you don't get your cadence right, then it all went murky.

What are they main differences between the theory behind Nexus and the practical application?

I think everything the book says versus what we have now is very different. We are not implementing those things properly, like I just mentioned now. All we are really doing now is, we are a whole herd of horses running for the finish line, hoping to get to the finish line at the same time. What I would have liked to have seen is…can I use an analogy? *Of course.* We call it the hamburger analogy. If you go to KFC and you order a burger, you do not get given the bun first to eat and then the patty, tomato, lettuce, etc. You get given the entire burger. When you bite it, you are supposed to bite through all the layers of the burger. So when you are doing a Nexus integration, at the end of every sprint you are taking a bite of your whole burger and you are supposed to be able to see that bite you are taking, of all the layers. You are not just getting the bun. That was the biggest issue I had, was the discrepancy between how we were supposed to have done it (whole burger) versus the way we did it which was sometimes we gave you a bun and a lettuce. In the Nexus integration we were not communicating on the level of "How can we give a bite of a burger and give incremental bites until the burger is finished?", we were providing small parts of each element, hoping that by the end you have eaten the whole burger. That's what I wish we have done. *Okay, that makes sense. So, this next question is not related to the previous questions. So, looking that analogy, are we implementing Nexus?* At this moment. No. *So, you are saying that we started with Nexus strong in the beginning and then somewhere along the line it dissolved into what?* A watered-down approach to delivery with Stand-Ups. *And is it working for us?* We haven't met our deadline, so I don't think so. Honestly speaking, I feel that if the Scrum Masters had had more input with the Product Owner into how it should have been done and had been more disciplined in making sure that the hamburger's bites were planned and kept that cadence. I'm pretty sure we would have hit our deadline by now. I think the delay is because we are allowing goal posts to move, and we are not being disciplined in pausing and doing it right. So, what we are doing now is, each team is running toward a goal, we're hoping we all get over at the same time, and hoping that once we are over, the whole burger is there too. Which is really silly. We are just having daily Stand-Ups and, in those Stand-Ups, we are showing who is dependent on/blocking who. So that which we are implementing, we are implementing in silos. There is no Nexus-level, there are no discussions

taking place between the team members. *But isn't that integration not a communication of integration?* It is. *Because all the teams are sitting in that meeting.* That's correct. So, Scrum has four events; of the four we are doing one. *Which is the daily Stand-Up?* Which is the daily Stand-Up. Scrum is not effective if you only do one piece of it. That's not Scrum. The Nexus book and guide both say that this type of implementation, if done like this, with these events, is Nexus. Any deviation from this, you are not doing Nexus. It is literally a sentence in the Nexus guide. So, my answer to you is, when you do it right, it works. When you deviate, you are not doing it right. We've deviated to a point where we are running in silos; all of our teams are either doing some version of, either Scrum or Kanban, getting somewhere to something and hoping that's enough. *So, we've almost gone from non-scaled to scaled and then back to non-scaled?* Yes. But now all that we have got is visibility during a daily scrum that says where there is a block or dependency.

If you knew what you knew now, would you do Nexus again?

Yes.

What would you change, if anything?

I still believe, one hundred percent, that Nexus was the right thing to do. It was the only scaled process that I know of that is one product-several teams. Other Scrum frameworks are several products-several more teams. So, I would definitely still choose this. It's one product; we are building one burger, not several of them. I would have approached my relationship with the Product Owner and stakeholders differently in the beginning and let them understand the buy-in, let them understand the value of it. Perhaps they didn't understand the value of the process we were implementing, maybe coaching wasn't executed properly and that's why it's never been supported.

In general, what is your personal opinion on scaled-Scrum?

Amazingly so, the person who wrote the LeSS book writes this all the time; it says, "People think it doesn't work, but it works." Where you have the buy-in of all the stakeholders and you have the absolute dedication of everyone else involved, it will work. I have not had anyone in our team telling me that Nexus is not working for them; it's not the Nexus, it's the way it is implemented. They all agree that we must scale, and we have to scale, there has to be a way to scale. Of all the ways, this is the one we found the best. So, my personal opinion is it is pretty cool, I am a very big fan of Nexus. *I'm not just talking about Nexus alone though; I'm talking about the whole idea of scaling scrum?* I have had individuals from other companies saying, "Can we please have you come and explain this to us because we think it will work for us." So, I think people out there are also seeing that they need to implement something similar,

especially on single product builds because you have one product, several teams; how do you get them all together? That's the point, is getting them together. I don't know how about SAFe, I don't know about LeSS, I don't know about DAD; but definitely a framework that I think would work well is Nexus. I think that you could even do several Nexus' in a company. So, say for example you have a mega-company and one set of five teams are building a product and another set of five teams are building another product, each one could have their own Nexus integration team. *But then there would have to be integration between those products?* But why? Say there were totally different products. *But if there were joins between the products?* Then you might consider SAFe because has got different rules guiding various teams building various products, building towards something. But Nexus is one product, several teams. That's it. My personal opinion is there has to be something that works at scale, and this is the newest thing on the market and we tried it. If everybody tried the thing that thought was going to work and it worked the first time, then we wouldn't have these people that had this massive determination in their lives, usually the fifth business works; the sixtieth time of Instagram is when it worked the first time best. Everything has iterations and iterations.

e. Transcript of interview with Scrum Master 2

i. Pre-Nexus

In your opinion, what are the main differences between Scrum and agile?

Simply put, agile is a state of mind, it's a mindset. It's not a framework, it's not a methodology; it's simply a way to think about things and that's why it can be applied to many things and still work. You can apply agile in your personal life, you can apply agile to projects that you do, and you can apply an agile mindset to the workplace. A mindset is something that isn't tangible, it's obscure, and that was the problem. People needed a way to, not just understand agile, but to actually become agile and gain that mindset. What then happened is, quite a few frameworks were developed to help provide the actions in becoming agile, if that makes sense. Agile then is a state of mind, whereas Scrum (and all the other frameworks) are basically like vehicles that make the values and principles of the agile manifesto more tangible, more attainable. *With that in mind, what other frameworks exist?* You have got Kanban, Crystal, a methodology called XP (Extreme Programming); those are the popular ones. You obviously then have Scrumban (which is a combination of Scrum and Kanban), I do not recommend it. Then you have stuff like test-driven development, which are agile-related practices which come from Extreme Programming. Those are just a few of the frameworks.

What was Hello Group using before they realized that they needed this scaled Scrum approach?

Okay, so I'll rewind a bit to a bit further before Scrum; before that we did Waterfall. *For timeline sake, how long ago was that approximately?* About a year and eight months plus; almost two years ago. We were using Waterfall and we were attempting to move to Scrum and become agile, but we were very selective. The people working here at that time didn't want that friction and pain that comes with implementing a framework, they wanted to pick and choose components of the framework to use those. The thing is, when you do that, you require a collection of processes and practices to change behaviours and, if you change behaviours over time, you change habits and you change mindset. Through changing a mindset, you change a culture. That's what was lacking here, they were picking some of the easier things to implement from Scrum, and from other agile frameworks and they were kind of very loosely implementing it and then hoping to see the benefits. We then moved to Scrum officially about a year and eight months ago. We moved to Scrum because the workplace changed, the world we work in changed. The traditional industrial paradigm was based on complicated work. That means that you have unknown knowns, there are knowns, but they are unknown, you can figure them out. This is complexity science. The software development workplace functions in the complex domain, which is the second last domain. The industrial paradigm functioned in the complicated domain, now we take a step up in software dev where you are in the complex domain. What that means is it is the realm of unknowns. You don't know what you don't know. Where the first paradigm made sense is there were unknown knowns, so you could apply a framework to a complicated environment and figure out what you don't know because things were relatively slow moving, not so interconnected and weren't as fast as they are now. What happened in between is that, with the information age came interconnectivity and speed; both those factors together meant that change compounded. Suddenly decisions needed to be made quicker, so now we decided to sacrifice efficiency which was one of the key factors of the industrial paradigm for agility. Agility means to be nimble and to respond quickly to change. That's where Scrum comes in. Scrum allows you to decentralise decision-making or authority and it allows you to plan incrementally and not plan up front because in a complex world, things change very quickly. So, you plan just enough so you eliminate waste. We applied Scrum, and the reason we picked just Scrum and not a scaled version was because we had teams that were their own little ecosystems and they worked on their own features, and they could almost build that entire feature without collaborating with another team. They had all the cross-functional skills required to build and increment. We applied Scrum for about a year, then with Bank, we realised that we are going to have several teams that build the same product and that have to build the same features so the change from Scrum to scaled Scrum was separate teams in their own ecosystems, fully capable of building a feature, to having to build one big product with three plus teams. If you have three to nine teams, you start using Nexus, if they are interdependent.

With the Scrum system, what was working for the Group?

A lot was working actually, we reduced risk and exposure to the market. What happened is, before Scrum, we would plan something for months beforehand and we would start implementing it and would've wasted three to six months on building a plan and then implement the plan only to find out on day one that the plan changed. That would change every single other part of the plan as well, it had a domino effect. You would then waste all that time out of the market while you had to plan, and then that plan would become redundant as soon as you implemented it. What Scrum allowed us to do was accept that we work in a modern world, a different world to what Waterfall solved, so we need to adopt a new paradigm. We knew that Scrum would allow us to plan just in time; so just enough planning in front so that it doesn't become too vague and then implement that mini-plan and during that implementation, extend that plan for the next cycle so that you're incrementally planning and working. Another thing that worked was the teams began to collaborate a lot better because they were now cross-functional so there were no handovers and everyone had context of what everyone else was doing so they could make the decisions for the entire product, not just for their function in that product.

What didn't work when it came to Scrum?

What didn't work was that we realised that Scrum, although many people think it's a team-based thing, it's actually a systems-based thing. If you fixed the team, or you improve them with Scrum, you're going to hit a glass ceiling at some point because you can only optimise them so much before they organisational gravity pulls them back into bad habits or impacts their productivity. So, we optimised the team and once the team reached a certain level of productivity, we realised that factors outside of the team started impacting the team so that they couldn't grow more. They reached an optimum productivity level and then it kind of dipped because there wasn't any motivation. That was one of the big things we had to fix. Scrum wasn't only a localised team thing; it was a much bigger problem. If you wanted to fully tap into the power and benefits of Scrum, you needed to look at the system as well, to look at bottlenecks before and after your function. There is a book that talks about systems and it says that any attention given to any area away from the bottleneck is an illusion. If you focus on improving the Scrum team and that's what we did, we gained some benefits, but it was an illusion because we didn't focus on the bottlenecks before and after the team. In terms of the entire production line, the entire value stream, where work comes from and where it goes after the team is done with it; you need to focus on that. We needed to resolve the bottlenecks. *How did you guys face those challenges?* There were a few steps that we took where we looked at where the team currently is in terms of their approach and where the team ideally needs to be. We realised that

where they need to be included a lot of the functions that were outside of that team at that current time, for example testing. So, the team wasn't quite cross-functional, there weren't testers sitting inside the teams, they were sitting in their own team. So, there was still a mini-handover, a mini-Waterfall inside Scrum. We listed this as an impediment and shifted our focus into solving this bottleneck and realised that we needed to break down the walls between these two silos and pull the testers into the teams to make it fully cross-functional. This, once again, reduced time to market and reduced work because now we had a better understanding of the work because they were sitting with the guys who were developing it, they were gaining context as it was developed. That's one of the ways we moved to systems-thinking rather than just thinking about implementing Scrum on a local team level. In a case of, stuff that impacted us before the work arrived at the Scrum team again, it was stuff like how the work was prioritised by EXCO. So, again we realised that they don't have any specific framework for prioritising and channelling work into the value stream, so we needed to set up events with them to help them scale their priorities and then audit these priorities so that there was a nice flow into the team. *Okay, and that would have been the product backlog review?* So that was additional events that we called the leadership events and the leadership board. The leadership board was an overarching board that was placed just before the product backlog where all our initiatives for the next couple of years would be listed and recorded. This gave us an additional redundant step because we were just creating silos again. It was still good because we realised that we needed this. We then created one product backlog for the Bank now and that is taken to EXCO and they prioritise on that with the Product Owners.

What is your version of a euphoric work setup in terms of teams, leaders, etc. and how does that differ from what is available in the Hello Group?

My euphoric work environment is an environment where every employee is completely autonomous. I think that that's the overarching goal of every Scrum Master or should be. Giving people the tools to decide how to do their work on their own. They shouldn't have any direction in terms of how to do their work from superiors or line managers. So, for me the ideal workplace would be a workplace with no hierarchical work structures and no siloed structures, a mesh basically. Instead of having this sequential, horizontal, and vertical ladder or frame, you have teams collaborating together without hierarchies and people deciding how to do their work and people been given objectives not been given actions. There should be shared consciousness across all business functions, everybody should have some sense of how people work in the other functions and why, that would make them better at doing their own job because they see the system. At Hello Group, we have gained a lot of that but are still in our transition. We are not focusing on implementing Scrum, we are focusing on changing the system. I would say we

are eighty percent there, although we will never have an end state, there will always be something new to do. At Hello Group though, in terms of our initial goal, we are eighty percent there because we have decentralised decision-making authority, so the teams are able to be given an objective and they are allowed to best decide how to achieve that objective. Hierarchies in teams have been broken down so, yes, we have Tech Leads, but they are not a managerial position, it's more of a servant-leader position, a facilitation role. They don't provide actions, they don't provide directions, they are simply there for support. We have got that now, however, beyond the team level there is still some hierarchy. So outside of the team, you still have a line manger to report to above the team, so that's one thing that we still need to focus on breaking down further and flattening it out. The issue here is that, if we want to be agile and we need to make a decision now because something has changed in the market, we need to capitalise on that change. Right now, we need to push information up the chain of command and then wait for a response, a sign off. By the time this happens, the opportunity could potentially be gone. In traditional organisations, you are going to have a much longer chain of command, but here we have reduced that chain of command to two or three links maybe, which is a lot of progress, but we are still not completely flat yet.

What methods or frameworks of scaled Scrum were looked at?

We considered things like SAFe. SAFe is probably the most popular scaling framework for Scrum because it is more commercialised. Similar to the CSM and PSM; CSM is lower quality certification but more people have it because it is easier to get. PSM requires you to give it more attention in order to pass the certification, but less people have it because it is more difficult. The same applies for SAFe, because it is a commercial product, people don't often do the extra research, they would rather go with what everyone else is using. There is also something called LeSS. Their motto is "less is more". This is also a way of scaling Scrum, but it is not liked by a lot of the community because a lot of the ideas in LeSS is counter agile. Those were the two main things were considered along with Nexus.

Why was Nexus chosen? What factors made Nexus favourable?

It was simple. The reason I chose Nexus as the framework was because we had already implemented Scrum and the teams were already used to Scrum, they could run their events, maintain their artefacts, and make all the decisions themselves within the Scrum framework. So, they knew it really well. Nexus is the closest scaling framework to the original Scrum framework and as such it requires the least amount of additional processes. So we wanted to go with what was going to give us the least amount of friction in terms of scaling and also what was going to be the most familiar in order to decrease the learning curve. If you reduce all these things, if you can almost implement something and help people understand it without creating

too much friction, then they are more likely to adopt it and use it going forward. That was the key with Nexus, we knew that if we wanted to work ourselves out of jobs as Scrum Masters, we needed to give people something that they are familiar with and that they could implement and use with minimal change.

Do you have examples of where scrum at scale has been implemented successfully? If you can give them that would be great.

Yes, so I have implemented Scrum at scale in my previous organisation, it was a team of about 200 developers; but I did not implement Nexus or SAFE it was literally extending the whole Scrum framework, which is Nexus but I didn't know it back then, because Nexus is relatively new. There are many examples of where people have used scaled scrum, Microsoft uses it to work with like 2500 teams across the globe at once. Microsoft uses it, Google uses some forms of scaled scrum, Cathay Pacific uses Nexus. The FBI implemented scaled scrum as well, there's a lot of information that fell into gaps between certain American governmental sectors, you had the military, the FBI, you had some other secret service agencies that all bits of information, they all has bits of the puzzle but because they didn't collaborate stuff fell through and something like 9/11 happened. Every single bit of information they needed to know that 9/11 was going to happen, everything, they all had collectively but no one put the puzzle together because they didn't collaborate. They all felt they were the better organisation and that they don't' share information, then 9/11 happened and they did some digging and analysis, and they realised that every single bit of information that could have pointed to what was happening was captured by separate divisions and separate organisations but they just didn't bring them together. So they decided to bring them together to actually implement scrum at scale, and since then, apparently they've seen a big increase in productivity and preventative measures against terrorism. The US military did a project in 2003, where they took 7500 people across the globe and they created a big team consisting of Seals, army rangers, of FBI, of homeland security; 7500 people that they put together into this one team that was headed by a guy called General McCrystal, he stumbles onto agile and Scrum and he implemented Scale. He was facing a problem where usually you fought a war against a known enemy, Germany against the US etc. they are also dressed up in military clothing. With Al-Qaeda, or the newer threats, you couldn't say who was the who. They could make a decision to bomb a city within a few minutes of having done any other planning, because any specific member of Al-Qaeda could make that decision to do something. And the hierarchical structures in the US military, the communication up and down the chain of command took time, so they couldn't respond. So he was like what can they do, he realised that the internet connectively and the speed of information is resulting in them working quicker than us; the decentralised decision making, the shared consciousness

is making them quicker than us, let's take that and restructure ourselves so we can be more effective. That is one of the main reasons there has been a down-turn in the terror attacks recently.

ii. Post-Nexus

When you had decided to go Nexus, how was this proposed to the company?

I was requested to come up with a solution or framework. *Specific for bank?* Yes, specific for bank because that was the first time that we needed to apply Scrum at scale. I'm a big supporter of Scrum.org and did their PSMs and knew about Nexus and I liked what I read in the Nexus case studies and have applied Nexus at my previous organisation. So, I had some familiarity with Nexus, although wasn't opposed to implementing SAFe; I was just more familiar with Nexus. So, what I did is have several meetings which included the teams, because remember we should not direct and tell people how to do their work, we should include them in the decision making and that's what we did. We provided them with all the options and said they can use Kanban, SAFe, or Nexus and these are the differences between them all. We then asked them, grounded in the context of their work, which one they felt was best. All answers kind of pointed towards Nexus. So, it was bottom-up intelligence, we didn't decide on behalf of everyone, we put our money where our mouth was and decided that, if we are going to practice what we preach, we can't tell people what to do and how do it, we have to trust that they have enough knowledge to help us make this decision. We took it to them and guided them through the process and figured out together that Nexus was the best solution for this scenario.

How well did the adoption of the new system go within the company, teams and people?

Very well, in fact I think that Nexus was adopted quicker and more successfully, in a given time, than what Scrum was. *Why do you think that is?* Okay, it was because we did it right. We didn't force people into using a framework, we included them in the decision-making process, so we decentralised authority and asked them for help in making the decision. They are the closest to the work and they understand the nature of their work, so they were best positioned to help us make the best decision for their situation. That's one thing; the other thing is, because Scrum was already implemented successfully, it was easier for the teams to understand Nexus, it was a very small learning curve. John Gold, a systems expert, he wrote that: "A complex system that works almost invariably evolves from a simple system that worked." Meaning, start small, build a foundation, and then scale on top of that. If you are going to build a complex system from the get-go, you are going to struggle and it's going to take you a lot longer than you planned.

Within Nexus, what has worked and why?

Okay so, I'm going to tell you a little story. This is to give you some context because I can tell you all the "robot quote out of the guide stuff", but that's not going to work. So in 1805 there was an admiral called Horatio Nelson, he was a British Navy Admiral; he was preparing to go into battle with the Franco Spaniards, which was Napoleon's army. The approach before that, was to approach the opposing navy as a unified front, so you would form this horizontal line and you would actually approach them that way because that meant that the length of your ship was facing open towards the enemy which means you had full vision of them. Also, the cannons at that time were built along the length of the ship so you had to move your ship that way. So, what Nelson said was that we have been doing this same strategy for thousands of years, we have been approaching each other as a unified front, so that means that the enemy knows exactly how we are going to approach and I know how they are going to approach because that's just how its done. So, he came up with this thing, and it has two elements. He said that everyone in his fleet; every person, every captain, every person on his ships needed to have shared consciousness, everyone needed to understand the entire system. What he did before this battle is, he educated everyone in war tactics and everything they needed to know and buy doing that he created transparency between the entire crew; that was step one. The second thing he did is he said that they needed to extend the decision-making authority to empower execution. We need to empower every single person to execute without a command, and that was controversial that that time because in those times, similar to traditional businesses today, you have one person that makes all the decisions and gives the orders, and that's what Napoleon's guys did. The middle ship had a flag and they would raise that flag to give orders, so everyone would watch out for that flag and if they saw the flag, they would execute the action. So, the problem there is that, if that ship goes down, there is no flag; or if any of the other ships move slightly forward or backward then they won't be able to see the flag. This delays decision-making and creates confusion and a delay in today's world is not a good thing. That is centralised decision-making. What Nelson did is understood the shortfalls in this method of attack, so what he did was that he decided to go in perpendicularly (vertically) to Napoleon's fleet rather than a unified front. Napoleon had thirty-three ships and Nelson had just twenty-seven, so he was clearly outnumbered. He created two rows of ships, then sailed towards Napoleon's fleet. They already have a smaller target area and that's a good thing because the cannons were fairly stationary and rigid. They then sailed all the way to Napoleon and they completely blocked Napoleon's middle ship off from the other ships in the fleet, no one could then see the flag and those ships couldn't fire in the way that they were used to, this rendered them useless, they could only shoot forward. Because Napoleon didn't share the decision-making authority, and because they didn't create transparency and create context of the system, these guys were too afraid to take action. Nelson on the other hand had taught everyone how to make decisions during a battle, and so

they knew that they did not have to wait for Nelson's orders (flag) and could do what they thought was best in that current situation. That's exactly what they did, and they destroyed Napoleon's fleet, despite being down on numbers. They won the battle and took some of Napoleon's ships back with them. So, in summary, Nelson created shared consciousness by showing everyone the system, giving everyone context and creating extreme transparency. He also extended the decision-making authority. That's what it boils down to with Nexus, is if you want to get the most out of Nexus, you only need to think about how to extend decision-making authority to other people so that I don't hoard decisions and knowledge to myself, number one. Number two, how do I create transparency through shared consciousness? That's what worked with Nexus here, we have allowed people to be fairly autonomous and to make decisions for themselves. Why? Because they understand the system as a whole.

And then what didn't work?

Well, nothing did not work. Some things didn't work as well as we thought it would, but that was a learning curve; we inspected and adapted. So, Scrum and Nexus aren't solution-providing frameworks, they are impediment-exposing frameworks. They shouldn't be looked at as a silver bullet that is going to solve your problems, they should look at it as a thing that will shine a light on every single issue that we have so that we can address it head-on and improve. In terms of what didn't work initially; the first thing it exposed was that we were working together as eight teams on one product and there were a ton of dependencies between the teams. In retrospect, there was a different approach we could have taken, we could have said to the people to self-organise themselves into efficient teams, let's look at the functions we need to build and let's make sure we organise ourselves into teams to reduce dependencies. So, we shouldn't have look at it in a way where one team is in India, therefore they work on this one function; this team is in Mauritius, they work on one function; this team is in South Africa, they work on this one function. We should have eliminated the function and geographic side of things and rather looked at what features there are and let teams, based on their skills, form around the feature to eliminate dependencies and the constant back-and-forth. This is why we implement forums to create transparency like the Nexus Stand-Up, so that on a daily basis, at least every twenty-four hours, people can communicate and resolve dependencies.

From the initial implementation to today, what are the things that changed and what was adapted?

Nexus remained exactly the same, it's a framework. A framework doesn't give you a step-by-step sequence of actions to get a desired outcome, that is a methodology. Nexus is a framework which provides you with the boundaries to work within; through its artefacts, through its accountability (such as Product Owner, dev team, Scrum Master, etc.), through the events,

creates a boundary within which the teams can self-organise. So, Nexus didn't adapt a lot during the course of the bank project but what we plugged into Nexus is what changed. For instance, the Nexus daily Stand-Up still remained, so that was a boundary that it gave us; every twenty-four hours, we had a feedback loop in the Nexus Stand-Up. What happened within the Nexus Stand-Up, how it was applied, had to change. We adopted the same way we did it in the normal Scrum framework, it wasn't a status update, it was more of a collaborative discussion around the goal; what did I do yesterday, what am I going to do today, what are my blockers? That just didn't work at a Nexus level because that wasn't the bottleneck or the problem, the problem was that there were teams that were interdependent, that needed to build features together and were simply waiting on one another, there was a lot of obscurity and grey areas as a result. We needed to reduce that, and the Nexus daily was the perfect opportunity. We then changed the approach in the Stand-Up and decided to focus on DIP, which is an acronym we came up with, which stand for Dependencies, Integration and Progress. In doing this, we place dependencies and integration issues in the face of every single Dev, every single day; they cannot avoid it and they don't have any lack of clarity. This made any complexities more transparent and that meant that the teams could do something about them, collaboratively. That's one way we learned from Nexus. Remember, like I said, a framework is something that doesn't give you end-to-end solutions, it gives you boundaries and your job is to implement that framework as is and then to plug in other processes and practices, like test-driven development, other methods of running methods into the framework to optimise it. Therefore, the framework can remain more or less rigid. *What are the main differences between the theoretical and practical application of Nexus?* So, you have to remember that Nexus wasn't, well Scrum wasn't created based on an idea that someone had, and then they kind-of popularized it, and then they said it has to be implemented. Scrum was formed from experience, was actually guys working on the ground, working in the trenches, figuring things out and then deciding that there's a better way to do this; then implementing that better way of doing things and testing it, so validating it, and then that became Scrum. So, Scrum came from practical application and solutions to challenges, so not a lot has changed. We implemented Nexus as is and it didn't change a lot; if I'm being honest, we didn't change much, it just remained Nexus. That is what a lot of people forget about Nexus and Scrum, is that they always say to you; 'this is the theoretical side of things, theory doesn't always work in practice', and they're right, it doesn't but Scrum wasn't theory first it was actually practiced first, for like 6 years, before it was even documented. The only thing I can say, where it's not implemented in the way that it's envisioned in the guide; is when you implement something cold-turkey, like you go all in and you just implemented end-to-end, completely, you're going to give people a shock because you're going to ask a lot of people, it asks them to change their mind set, which was deeply ingrained in them for years and years and

years. If you do that you're asking about because you're going to give people a shock and they are going to be unhappy, so we took more of a transition to implementation.

What factors do you think, if changed, would have helped the some of the issues that we had?

Again, if teams weren't formed based on geographical location. That's a big one because we chose to go with teams centred around their geographical location, we created more dependencies. If we could have helped the teams self-organise into teams around features, according to skill sets of employees regardless of location across the globe we would have reduced dependencies. The trade-off there would have been that we had geographically distributed teams which needed to dial in one another to have meetings and stuff, that is operational overhead, so there was a trade-off there. That's one of the things which could have changed, the other thing is that Nexus doesn't give you, and Scrum, it's a framework so it doesn't specifically give you actions and so a lot of the stuff that we did basically was pure Nexus and Scrum, so it was just a framework and we needed to figure of what processes and tools apply within this framework to optimise it; however, had we had a sort-of implementation plan almost, of at least recommended steps. So, like create transparency with information radiators, like television screens that radiate progress on the board. If we knew of that before the time, or we had a little checklist of best practises that would have reduced the time it took to actually get the ball rolling.

What would you say are the biggest findings/lessons that we learnt from implementing Nexus and this build?

I think the biggest thing we learnt, was that in order to be nimbler you need to give everyone in every single team shared consciousness. So, every person needs to understand the entire system, to an extent, they don't have to be an expert with complete knowledge about something, but they need to have deep knowledge about their domain and then breadth of knowledge across other domains. The biggest learning was that you need to create shared consciousness from the beginning, you get everyone to understand the entire system to a certain extent. If you get people to do that, and there are various ways to do this; you can depend on people to make the correct decisions themselves, you can trust them. You can say, 'Guys, you understand the system now, because you understand the system now, you can make a better decision'. So think about this, imagine a sports team and say it's a soccer team, and each player on that field has a different manager or coach, that person individually coached and he's coached really well; he has the skills Like if he has the ball on his own, he can lie of his back and do all these tricks, he's amazing in terms of his skill set; and so is every single other member on that team, because they all have individual coaches focusing on just them. Suddenly you throw them onto a field and have them play against another team that was coached together, the team the we're team

that we're referring to is the one that has individual coaches, definitely has the better skillset because they got that individual focus; but do you think they are going to beat that other team? No, they haven't worked together, and they were completely focused on what they were coached, which didn't express anything of their counterpart's role. So, they don't know what is this guy going to do next to me, is he going to run forward, is he going to run back if he going to stand where he is, what's the deal? Because they don't have shared consciousness, they're not going to be a good team, they are not going to be able to I had a ton of individuals that had super great skill and they could all build amazing things but they didn't have shared consciousness, so they couldn't make a decision that was best of the entire system, at the beginning. We took a while to learn from that, that is why we had to bring the dependency and integrations into the Nexus daily Scrum approach, because saying, 'what I did today' and 'what I did yesterday' and my impediments, but rather what was going to make us grow as an entire team, was going to help us unblock individuals. If we had learnt that earlier, we would have been much better off from the beginning. So it's creating shared consciousness and decentralising decision making authority.

If you knew everything you know now, would you do Nexus again? If so, what would you change, from implementation to general running? If not, what would you consider implementing instead?

I believe in a fit-for-purpose solution, so I wouldn't say I will always opt to do Nexus, but for similar situations it's going to be very hard to get me not to do Nexus. This is because it works, it's simple, it's effective and it's quite rigid and it gets you to gain an agile state of mind, and that's what your goal is. I wouldn't really opt for anything else in the same situation but as I said I believe in fit-for-purpose, so where something in the nature of the work differs I would definitely want to do go with Kanban for something that is more adhoc or based on flow of work.

So, in terms of the banking project, this build, would you consider Nexus a success?

Yes, Nexus was definitely a success. If you look at other companies they have had similar projects with a lot more money and time, we've been doing Nexus for 8 months and we've achieved the same thing, at a fraction of the cost. Nexus was a success, whether the people here would tell you it was is a different story. The thing is, and that unfortunately that is the situation, is that no one knows what would have happened if we didn't implement Nexus. They only know Nexus, they only know what we've done up until now with Nexus, so we can only look outside of our environment to people that have done things the Waterfall way or maybe a different framework and we can compare results. Did we build more-or-less the same thing as them, the same complexity, the same amount of people, and what was the time differences in

costs. And if you look at it, we got more done, with less people and less money than the other companies; so that's why I would say Nexus is a success.

What is your personal opinion of scaled-Scrum?

It's easier said than done and you need to make sure that you're up for it. You implement scaled Scrum you're playing with people's mindsets and you're playing with behaviours. It's not the simple thing of 'I can do it and forget about it later'. So, my opinion is that it's very very powerful, but like they say a powerful gun gives you the ability to shoot yourself in the foot powerfully. The stronger the framework and the benefits, the stronger the potential downside is as well. My opinion is that it's worth it if you're willing to invest the time and willing to invest lots of hours and research and reading, and you're passionate about helping people and teams succeed; and because of that a consequence is helping businesses succeed. But if you're not very passionate and you're not going to put in the time, then it's not for you. It's an amazing tool but people need to use it correctly and put in the time.

f. Transcript of interview with Scrum Master 3

i. Post-Nexus

What has worked and hasn't worked in terms of Nexus for this bank build, and why?

What has worked is having the teams collocated and having all of your teams which are dependent on one another and require cross-functionality between teams to sit at the same site; at the same time this hasn't worked as one of our major dependency teams is on a different continent. On the one hand we see the benefits of co-habitation of teams and how they manage to talk to each other and the way in which they are able to resolve their issues faster with having the product owner and their BAs on hand. On the other side we've seen the negative of this having a major dependency on another continent and seeing the extreme worst of that dependency. *Okay, what do you mean by 'extreme worst of that dependency'?* Okay, I'm going to call them 'the dependency', the team that we are dependent on is 5/6 hours ahead of us; by the time we are hitting our peak delivery and performance here around lunch time, they are getting ready to go home. When we need them, when we're in the midst of fixing issues, finding bugs, and resolving things, they are now not available to assist. Also, a big issue we are facing now, is the team which is away from the Product Owner and their direct BAs, very quickly lose sight of what is meant to be built. There's an example of one of the teams and the security issue, where things were discussed over and over again via Skype but without the Product Owner constantly being available and constantly being in a position to constantly answer questions, the team, and the team leads, and the Tech Leads lose vision of what they are actually supposed to be building. *And when you say, 'lose sight', I'm a assuming it causes derailment of the project*

or of that build? I wouldn't say necessarily derailment, that is definitely an extreme, but it would definitely cause delays. So however much communication you can managed to get through Slack and through Skype, and through regular phone calls throughout the day; having someone or the teams you are dependent on, literally sitting next to you, is the most efficient way to get work done. I'm not saying it's not possible sometimes your skillset has to be in another country, it just doesn't exist anywhere else. In this specific example, it would have been a lot easier if those skillsets, if all the teams involved, were collocated.

Keeping those points in mind, besides being collocated, are there any other factors which if changed, would have helped the issues brought up?

An example we experienced today was having the stakeholders be involved with demos more regularly. So not getting to a point where development has been going on for four to six months and end-user hasn't seen the product, and come basically go-live, the end-users actually see the product and raise a whole field of concerns. This doesn't directly link to scaling-scrum, but it would directly link to scrum, which is the foundation/fundamental part of scaling scrum. One of scrum's fundamentals is to have continuous feedback, have the end user and stakeholders ultimately signing-off and giving feedback, improving the product, and raising potential issues on a very regular basis This is something we have had to a certain extent but not to an end-user extent, we've had it to a business-partner extent, but not to and end-user extent; and that would have helped, I think even when we go-live we will see that. *So that is something else that hasn't worked?* Yes. I would also go into process, if you're going to do Nexus, and you're going as we did, as this company did; we went in as a very inexperienced, nobody involved in the Nexus implementation actually, practically had done Nexus before. Taking more time to understand Nexus, and the way it is meant work, and scenarios it performs most efficiently in, the kinds of problems it works in best; taking the time before you implement before you really probe people that have done it before and forums and meetups. Finding those people, asking their advice, throwing different scenarios at them, to try and figure out; 1) is Nexus for you, is it actually going to work, what elements are going to work, should you actually be building separate products instead of one product and have completely different team integrate? *But isn't the base of Nexus one product, many teams?* One product, many teams. So, if you look at bank now, there are essentially 5 products that sort-of bounce off each other. *But is bank not the one product and all those 5 things components of that one product, seeing as there is integration between all of them?* There is integration between all of them, but one does not necessarily require the other to function, it can be built in a way in which each can function on its own for the most part. The fundamentals of Nexus are built on the, well the example they use in the Nexus guide is a security system; where literally the camera has no function without the laser

trip and the motion sensors, and the motion sensors have no function without the alarm, and the alarm has no function without the doorbell with the camera on it. Where literally every single piece of the system is supposed to rely on one and other for their fundamental purpose. The analogy that comes to mind is a circus, the bank is the circus and the different elements we're building are the different shows in the circus, so it can stand on its own but it's not a circus without everything together.

What are the main learning and lessons taken out of the Nexus implementation within this product?

I think, because I joined very very late in the Nexus implementation, and I learnt very little from the handing over process, there were a lot of growing pains that I am not aware of. But from what I can see, Nexus is a framework it's meant to be adjusted; I think that sort-of encapsulates everything. Nexus is a framework, it's like Scrum, Scrum encourages you to stick to the rules initially and then deviate and adjust and bend the rules so to speak, to a point where it works for your organisation. Have your integration team understand their purpose from the very beginning. *When you say integration team, you're referring to the main overarching team?* Yes, there's the NIT, which is select members from different teams within banking which are meant to focus on integration, blockers, and dependencies. Having them understand, so this goes back to understanding Nexus properly at the get-go, having them as a NIT-team and an integration-team, makes the entire process of integration a lot smoother. If person from Team A understands fully that person from Team B cannot physically build their product without this thing that Team A is responsible for, it puts a lot more emphasis on them to get that thing done. If there is a misunderstanding between what Team B needs and a lack of understanding as to when they need it for and how important it is to the implementation, nobody is really talking to me. That communication, that understanding of the teams need to talk about these scenarios all the time, this is your purpose, this is to talk to each other, find the best ways to integrate and the fastest ways to integrate, which obviously meets getting rid of your dependencies as fast as possible. If the NIT-team understands it from the start, I think the integration would be a lot smoother and then, in-fact, your deliverables; that also goes back to understanding Nexus properly. Other than that, it's the stereotypical items which would come up from any post-mortem of a project; which are communication, understanding specs, getting clear business requirements; which goes back to getting your stakeholders involved. These are the things that go with basically any big project, you're going to have communication issues, clarity of spec issues and in this situation, you're going to have integration issues.

What are the main differences between the theoretical and practical application of Nexus?

Planning does not go as smoothly as Nexus thinks. Nexus assumes that you have the vision of what you want the project to be and you have clear designations for what each team is responsible for, and that each team is fully cross-functional and doesn't need external testers or external front-end developers. It assumes teams are cross-functional and it assumes that you're building a relatively well understood, single product; and it also assumes your teams are capable of doing planning all together. Our teams were not. An example given would be that when we put all the teams in the same room, the planning does not happen; you waste 6 hours or 40% of your developer's time. For our example, the proposed practical solution to this was to have cascaded planning, where a specific team plans what they are going to do and then the next team plans what they are going to do, etc. That was not ideal, where we experienced deviation from the norm, well expected theoretical norm. One I don't fully understand that well, because I have not actually seen it in action, so it's one thing reading how it's supposed to work and seeing how it's supposed to work; product backlog refinement just purely doesn't exist. *But why?* Because we 'don't have time', and yes, I say this in inverted commas, 'we don't have time'. We don't do PBR and as far as what I can understand it is a massive part of Nexus, it's a massive part of being able to map your dependencies. Why we don't do it, the general feedback I have got is that we just don't have time; which is fair-enough as we don't have time to spare. The theoretical knowledge that I have, seems to sort-of underpin that you must to PBR in order to have a successful Nexus integration; you have to be able to map your dependencies, etc. This I have not seem practically implemented, so I cannot tell you if we're doing it better, or if the outcome has been more successful on PBR or not, but I can just tell you that we do not do this. Retrospectives-wise, once again we don't really do them, we've done one that I know of. So practical versus theoretical on retrospectives, once again you'd have to do the retrospectives to see the benefits of it. Do I think that if we had implemented retros more religiously, things would have gotten better? Definitely. There are a lot of issues on the Nexus integration level that needs to have been dealt with needs to be resolved, and processes that could have been put in place to make integration considerably easier, that would have come in retro and didn't, because they didn't exist.

If you knew everything you know now, would you do Nexus again? If so, what would you change, from implementation to general running? If not, what would you consider implementing instead?

Yes, I would, I would take the effort to understand the product better, or the intended vision of the product better. I would choose it, simply because it is simple to implement and it's simple to understand, and you can bend the rules without breaking them. What I would change is, I would have a kick-off initially so that everyone understands what each other team is responsible

for, which I don't think we did initially. This would also get everyone on the same page, or agreement, of how we are going to integrate, how are we going to communicate with each other. At the moment, there are 15 different channels of communication for the same thing, the current scrum master, being myself, is half the time unsure of which is the head and which are the toes of the conversation; because things are just completely all over the place. If anyone was to join, a developer or tester, the banking project at this stage they would not know where to begin; they wouldn't be able to catch-up because there is no agreement on the best approach to take. Yeah, so what I would do is, do a kick-off, make everyone understand or get everyone on the same page about what all the other teams are doing, what they main sources of dependencies are going to be doing. So in our case, there is a team doing all the back end APIs; if you have a back end API in mind let them know ahead of time. Define clear channels of communication, just so that everyone can keep abreast of the same conversation at any time. I'd also, oddly enough be stricter about having certain things, like Retros; but it wouldn't be a case of you must Nexus level Retro because you are doing Nexus, it would a case of, we're doing team Retros because of what Scrum says. Scrum says we do Retro because sometimes you just need to either have a team-build or there are major issues you need to deal with. On the cases where you in the sprint and there is nothing major going on, then it can be 30 or 15 minutes, if it is that extreme, but that ability to check in and ask if everything is okay, does anything need to be addressed, is important, even if it is 15 minutes. I wouldn't necessarily afford Nexus level Retros to that degree, it would be a case of if anyone has seen anything going wrong or if we are falling behind for some unknown reason, let's call a Retro and try and figure this out. Are we deviating from our expected standards and are we losing track of where we though we are supposed to be. If we come to that retro and we are under the illusion that "Hey, life happens" and nothing we can fix has resulted in us being later than expected, then we carry on as normal. There is nothing you can fix, but at least you have checked in and made sure there is nothing you can fix. What I do like, that I don't think anyone intentionally did is, Nexus says that says that each team has a Scrum Master and Product Owner, while there may be a team, there is a Nexus level Scrum Master, there is not really a team Scrum Master. One person does not have the time to sit with five teams every day, I can't dedicate a full day to each team. I can't see one team for one day a week and then never again, it's not possible. What I do really like, and it is only being done with two teams is, there is a nexus product owner, and then there is a team product owner that sits with the nexus product owner, this frees up the Nexus Product Owner to focus on the bigger picture and it does not create favouritism toward that Product Owner to one team, it does not neglect any team. What happened here is that we had BA's that were helping our teams and inadvertently became their product owners, working with the Nexus Product Owner, but that is only happening with two of the five teams. I would try to make that

happen with all five teams and then have the Nexus Product Owner liaise with those 5 product owners. It would almost be like a starfish, with the Nexus product owner in the centre and the tentacles being the team product owners. The people in the centre have the umbrella view of what's going on and then through the umbrella pass the information on to the team product owners and scrum masters. So do I think Nexus was the ideal scaled-scrum for this company, in its purest form? No, not at all, we tried it and it didn't work, the productivity wasn't seen from doing pure Nexus; adapting it later on did show more results. Do I think another form of scaled Scrum would have working better, not one that I know of. Not a scaled Scrum that I'm aware of would have worked better, I do think Nexus was the right choice.

In general, what is your personal opinion on scaled scrum?

In general, there is a necessity for it. Nothing can be built with just one or two teams and, if it could, the time-to-market would not be viable for business. So, there is a necessity for it and scale Scrum in general, more specifically Nexus, has an advantage as it is considerably simpler than some of the other frameworks out there. Trying to coordinate a handful of teams is hard enough on its own without bringing in the whole organisational mindset into it. Nexus focuses and leans this all down to focusing purely on the product development and implementation. Theoretically, there is a necessity for it. Do I think it is a really exciting way to work in groups? Yes, definitely. Is it the future on how we are going to work on scale Scrum in general? I think it could be. I think it could have external applications, outside of IT too. In this implementation specifically, I think we did the best that we could. Considering the fact that we had geographically disbursed teams, I think the fact that we had inexperienced Scrum Masters and Product Owners; we didn't have a clear idea as to what we wanted Bank to pan out to, in the beginning. Considering all these factors, I generally believe we have done the best that we can. One thing I do think, in specifically team Retrospectives, even if it's fifteen minutes; in those fifteen minutes you agree to go and have a beer after work. The human side of things. There is a general requirement to get things out as fast as possible and Nexus or scale scrum are designed to address this as lean and efficient as possible, but the flip side of the coin is that people get seriously drained, they get worked to the bone. I think that in all the excitement and pressure, we lost sight of the fact that while business is impressed, we are pushing the development teams to the point that they can't breathe to get to market faster. Nexus and scale Scrum, if executed properly, gives us the opportunity to take fifteen minutes to give back to your people, to take care of your people. To tell them that they are leading the trend in this industry and you deserve to be rewarded for this, even if it's donuts at lunch. That human aspect, I believe, very often gets forgotten, not deliberately, but it does get forgotten.

Appendix B: Survey Questions and Answers

Please find the survey questions and answers below. The blue text represents the questions asked, the black text represents the answers to the questions survey taken by External Scrum Master.

Why do you think a scaled agile framework is needed in industry?

There are many instances where the size of the projects being initiated are too large to be done in an ad hoc, uncoordinated manner. Scaling agile for these projects becomes an imperative.

What scaled agile frameworks do you know of, or have you used? What factors are considered when choosing a framework, as in, what makes a certain framework better for one scenario compared to another framework for a different scenario?

There are several scaled agile frameworks (Disciplined Agile Delivery, SAFe, LeSS, Nexus) but I have only worked with two of these. I believe that organisations select the framework based on the recommendation of the agile consultant, rather than on the merits/suitability of the framework itself.

From any of your previous/current implementations of scaled agile frameworks, what were the issues and/or reasons which contributed to the need for a scaled solution; as well as what framework was chosen?

At one of the large banks in South Africa, a decision was made to implement a scaled agile framework (DAD) because of the size of the organisation and the complexity of many of the projects being undertaken.

From any of your previous/current implementations of scaled agile frameworks, was the implementation and running of scaled agile a success? If yes, or no, please give reasons. Do you know of any other examples where agile at scale has been implemented successfully? Can you give some examples?

In my experience the implementation of the scaled agile framework in this bank was not a success. The main contributing factor, I believe, was a lack of buy-in and support from management, and a lack of a coherent and supported roll-out. However, my current colleagues claim to have successfully introduced SAFe at another one of the large banks. But I was not involved in this.

From any previous/current implementations you were part of/know of, what were the main lessons/learnings taken from the implementation? What has and hasn't worked, why?

Getting the support and involvement of management is crucial to the successful implementation. Without this there is no momentum and no drive to implement any changes. If

teams adopt agile as a framework without the understanding and support of the organisation, structures and processes within the organisation will not change to support the implementation, and the initiative will struggle to survive – teams cannot work in isolation, it must be a system wide adoption.

Can you recall any obvious changes/adaptions from the theoretical framework to the way the framework was run in the practical environment? Please specify the adaptations per framework implemented, if you are speaking to more than one framework.

In my experience agile was introduced to all staff member and it was made mandatory to attend the bootcamp course, but the framework was not implemented on an organisational level. This left teams and individuals trying to follow agile principles without having the support of the organisation in which it needed to be scaled. Furthermore, there were no agile champions/coaches to help with roll-out of the framework, which left teams working "agile" in totally different and inconsistent ways. They did not follow any single or coordinated framework, and definitely did not follow the prescriptions of the scaled agile framework that was selected.

In general, what is your personal opinion of scaled agile?

I think that if properly implemented and supported (and given a chance) it will definitely benefit certain larger organisations. Many of the projects and initiatives will benefit from the agile framework, and with scaling can transform the way the organisation operates, to their own benefit

Appendix C: Structural Codebook

Interview Topic	Question Number	Structural Code Name	Structural Code Definition
Understanding agile Terms	1	AgileTerms	**Brief Description:** Difference between the terms 'agile' and 'Scrum'. **Full Definition:** A scrum master's opinion and explanation of the differences between the terminologies of 'agile' and 'Scrum' within the project management realm. **When to Use:** Use this code to capture the personal and factual opinions of the differences between 'agile' and 'Scrum', while interviewing a scrum master; as well as any facts linking 'agile' and 'Scrum' to each other. All of this as a result of a direct response to question 1 or any probing questions related to it.. **When Not to Use:** Do not use this code if the participant being interviewed is not a scrum master. Do not use this code for discussion of 'agile' or 'Scrum' that are not as a direct response to interview question number one or any other probing questions which followed within question 1.
Ideal versus Real Work Environment	2	IdealVsRealEnv	**Brief Description:** Ideal working environment verses current working environment. **Full Definition:** Participant's opinion/idea of what the ideal working environment would be, regarding roles and how they interact with each other; versus how the company is operating currently.

			When to Use: Use this code to capture the idea of the participant's perfect working environment, as well as how it differs from what they are working with currently; all while being as a result of directly answering question 2 or any probing questions related to it. **When Not to Use:** Do not use this code for discussion when ideas of ideals work environments and/or other work environments, as well as the current work environment, are mentioned not as a direct response to question 2 or any probing questions related to it.
Current Work Environment	3,4,5,6	CurrentWorkEnv	**NOTE:** This is a placeholder, do not use it was a means of coding. Questions 3, 4, 5, and 6 are linked to this placeholder but will be represented with individual structural codes that link to this placeholder. Any information associated with any of those four codes will be considered linked to this placeholder for analytical purposes.
Current Work Environment -- System Being Used	3	CurrentWorkEnv_SysUsed	**Brief Description:** Description of what the current system is within the company. **Full Definition:** Participant's description what system is currently being utilised with in the daily functioning of the company currently with respect to the project in question. **When to Use:** Use this code to capture any information regarding the systems used currently within the company in order to operate the way that they do. This code is to be used only as a result of a direct response to question 3 and related probing questions, as well as relating to the banking project.

			When Not to Use: Do not use this code for discussion of the systems used within the company currently when it is not a direct response to question 3 or any probing questions related to it.
Current Work Environment -- Positives	4	CurrentWorkEnv_Pros	**Brief Description:** The positives of the current work system. **Full Definition:** Participant's views on what is currently working well with the systems being used within the company regarding the banking project. **When to Use:** Use this code to capture the factors that work well within the company's systems currently. This code can also be used if there is an overlap with questions and probing questions related to questions 5 and 6, and positive factors/opinions are mentioned, in either questions or related probing questions of 4, 5, 6, and are associated with the banking project. **When Not to Use:** Do not use this code for discussion of the factors that do not work well within the company's systems currently and are as a result of a direct response to question 4 and its related probing questions.
Current Work Environment -- Negatives	5	CurrentWorkEnv_Cons	**Brief Description:** The negatives of the current work system. **Full Definition:** Participant's views on what is currently not working well with the systems being used within the company. **When to Use:** Use this code to capture the factors that did not work well within the company's systems currently. This code can also be used if there is an overlap with questions and probing

			questions related to questions 5 and 6, and negative factors/opinions are mentioned, in either questions or related probing questions of 4, 5, 6, and are associated with the banking project. **When Not to Use:** Do not use this code for discussion of the factors that do work well within the company's systems currently and are as a result of a direct response to question 5 and its related probing questions.
Current Work Environment -- Causes	6	CurrentWorkEnv_Causes	**Brief Description:** The reasons that the currently used system is not working. **Full Definition:** The participant's opinion of the causes/reasons the current system is not working for the company and project in question. **When to Use:** Use this code to capture the causes of the cons within the system currently being used in the banking project. The reasons behind the cons mentioned in question 5, as well as other cons that have been brought up in either question 4 and/or question 6; and any probing questions related to any of the three primary questions. Can include reasons for cons mentioned in questions 4 and 5, and their related probing questions. **When Not to Use:** Do not use this code for discussion of a pro or con brought up in questions, or their probing questions, 4, 5, or 6. These factors can be considered for structural codes CurrentWorkEnv_Pros and CurrentWorkEnv_Cons respectively.
Need for Solution from Stakeholders	7	StakeholdersNeed	**Brief Description:** The need for a solution by a stakeholder.

			Full Definition: Participant's opinion as to whether or not there is a need for a solution to issues currently faced within the banking project. **When to Use:** Use this code to capture any needs of a solution to current issues faced within the banking project, when interviewing a stakeholder and a direct response is given to question 7 and any related probing questions, or if there is mention for need for a solution due to a negative factor brought up in the Current Work Environment subsections. **When Not to Use:** Do not use this code for discussion of needs for a solution a stakeholder wants if the participant being interviewed is not a stakeholder; or if the reponse is not related to question 7 or any Current Work Environment subsections.
Scaled-Scrum Solution	8,9,10	ScaledSol	**NOTE:** This is a placeholder, do not use it was a means of coding. Questions 8, 9, and 10 are linked to this placeholder but will be represented with individual structural codes that link to this placeholder. Any information associated with any of those three codes will be considered linked to this placeholder for analytical purposes.
Scaled-Scrum Solution -- Causes	8	ScaledSol_Causes	**Brief Description:** The factors which caused consideration for a scaled-scrum method. **Full Definition:** Participant's view on the reasons as to why the company considered a scaled-scrum methodology for the banking project. **When to Use:** Use this code to capture what caused the company to consider using a methodology such as scaled-scrum when dealing with the banking project. Use this code when the

			participant is directly responding to question 8 or any probing questions related to it. **When Not to Use:** Do not use this code for discussion of scaled-scrum causes which are not as a result of a question 8 response.
Scaled-Scrum Solution -- Methods	9	ScaledSol_Methods	**Brief Description:** Methods of scaled-scrum considered. **Full Definition:** Any methods of scaled-scrum that were considered with respect to the bank project. **When to Use:** Use this code to capture any scaled-scrum methods that are mentioned by the participants with regards to the banking project and as a direct response to question 9 as well as it's related probing questions. **When Not to Use:** Do not use this code for discussion if scaled-scrum methods are mentioned in a direct reponse to any questions other than question 9 as well as it's related probing questions.
Scaled-Scrum Solution -- Examples	10	ScaledSol_Examples	**Brief Description:** Examples of scaled-scrum methods. **Full Definition:** Examples of scaled-scrum methodologies being used in the industry. Practical scenarios and any factors or stories related to these cases. **When to Use:** Use this code to capture any examples the participant may know of which related to a scaled-scrum approach being used in a real industrial scenario. Consider answers which are directly linked to question 10 and/or it's probing questions.

			When Not to Use: Do not use this code for discussion of real-life scaled-scrum examples which are not a result of question 10 and/or its probing questions.

Appendix D: Quality Assessment Scoring

Source Identifier	Article Reference	Q1	Q2	Q3	Q4	Quality Score
#1	Denning, S. (2015). Agile: it's time to put it to use to manage business complexity. Strategy & Leadership, 43(5), 10-17. doi:10.1108/SL-07-2015-0057	1	1	1	0	**3**
#2	Denning, S. (2017). The age of Agile. Strategy & Leadership, 45(1), 3-10. doi:10.1108/SL-12-2016-0086	1	1	0	0	**2**
#3	Ebert, C., & Paasivaara, M. (2017). Scaling Agile. IEEE Software, 34(6). doi:10.1109/MS.2017.4121226	1	1	1	1	**4**
#4	Cohn, M. (2010). Succeeding with Agile. United States of America: Person Education, Inc.	1	1	1	1	**4**
#5	Reifer, D. J., Maurer, F., & Erdogmus, H. (2003). Scaling agile methods. IEEE Software, 20(4). doi:10.1109/MS.2003.1207448	1	1	0	0	**2**
#6	Prikladnicki, R., Lassenius, C., Tian, E., & Carver, J.C. (2016). Trends in Agile: Perspectives from the Practitioners. IEEE Software, 33(6). doi:10.1109/MS.2016.152	0.5	0.5	1	0	**2**
#7	Paasivaara, M., & Lassenius, C. (2014, 28 July-1 Aug. 2014). Deepening Our Understanding of Communities of Practice in Large-Scale Agile Development. Paper presented at the 2014 Agile Conference.	1	1	1	1	**4**
#8	Dikert, K., Paasivaara, M., & Lassenius, C. (2016). Challenges and success factors for large-scale agile transformations: A systematic literature review.	1	1	1	0	**3**

Source Identifier	Article Reference	Q1	Q2	Q3	Q4	Quality Score
	The Journal of Systems & Software, 119, 87-108. doi:10.1016/j.jss.2016.06.013					
#9	Bass, J. M. (2016). Artefacts and agile method tailoring in large-scale offshore software development programmes. Information and Software Technology, 75, 1-16. doi:10.1016/j.infsof.2016.03.001	1	1	1	1	**4**
#10	Fourie, L., & De Vries, M. (2017). Exploring enhancements to the agile approach for mid-sized enterprises in the services sector. South African Journal of Industrial Engineering, 28(3), 29-39. doi:10.7166/28-3-1837	1	1	0	0	**2**
#11	Conboy, K., & Carroll, N. (2019). Implementing Large-Scale Agile Frameworks: Challenges and Recommendations. IEEE Software, 36(2), 44-50.	1	1	1	0.5	**3.5**
#12	Schwaber, K. (2018). Nexus™ Guide The Definitive Guide to scaling Scrum with Nexus: The Rules of the Game (pp. 11). Retrieved from https://scrumorg-website-prod.s3.amazonaws.com/drupal/2018-01/2018-Nexus-Guide-English_0.pdf?nexus-file=https%3A%2F%2Fscrumorg-website-prod.s3.amazonaws.com%2Fdrupal%2F2018-01%2F2018-Nexus-Guide-English_0.pdf(Schwaber, 2018)	1	1	1	1	**4**
#13	Agile, S. (2018). SAFe® 4.6 Introduction Overview of the Scaled Agile Framework® for Lean Enterprises. Retrieved from	1	1	1	1	**4**

Source Identifier	Article Reference	Q1	Q2	Q3	Q4	Quality Score
	https://www.scaledagile.com/resources/safe-whitepaper/					
#14	M. Paasivaara et al., "Integrating Global Sites into the Agile and Lean Transformation at Ericsson," Proc. IEEE 8th Int'l Conf. Global Software Eng. (ICGSE 13), 2013, pp. 134–143.	1	1	1	1	**4**
#15	M. Paasivaara, "Adopting SAFe to Scale Agile in a Globally Distributed Organization," Proc. IEEE 12th Int'l Conf. Global Software Eng. (ICGSE 17), 2017, pp. 36–40.	1	1	1	1	**4**
#16	CollabNet, & VersionOne. (2018). The 12th State of Agile Report. Retrieved from https://agilebb.nl/wp-content/uploads/2018/04/versionone-12th-annual-state-of-agile-report.pdf	1	1	1	0.5	**3.5**
#17	M. Kalenda, "Scaling Agile Software Development in Large Organizations," master's thesis, Faculty of Informatics, Masaryk Univ., 2017; is.muni.cz/th/410499/fi_m/masters_thesis.pdf.	1	1	1	1	**4**
#18	Paasivaara, M., Lassenius, C., & Heikkilä, V. T. (2012, 20-21 Sept. 2012). Inter-team coordination in large-scale globally distributed scrum: Do Scrum-of-Scrums really work? Paper presented at the Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement.	1	1	1	1	**4**
#19	Dolman, R., & Spearman, S. (2017). ASK: AGILE SCALING KNOWLEDGE - THE MATRIX.	1	1	1	1	**4**

Source Identifier	Article Reference	Q1	Q2	Q3	Q4	Quality Score
	Retrieved from http://www.agilescaling.org/ask-matrix.html					
#20	B.V., T. L. C. (2019). Introduction to LeSS. Retrieved from https://less.works/less/framework/introduction.html	1	1	1	1	**4**
#21	Agile, D. Introduction to Disciplined Agile Delivery (DAD). Retrieved from http://disciplinedagiledelivery.com/introduction-to-dad/	1	1	1	1	**4**
#22	Francino, Y. Large-scale agile frameworks compared: SAFe vs DAD. Big agile. Retrieved from https://techbeacon.com/app-dev-testing/large-scale-agile-frameworks-compared-safe-vs-dad	1	1	1	1	**4**
#23	QASymphony. (2018). The Pros and Cons of the Scaled Agile Framework (SAFe). Agile. Retrieved from https://www.qasymphony.com/blog/pros-cons-scaled-agile-framework-safe/	1	1	1	1	**4**
#24	Scrum.org. (2017). Security Software Product Company Uses Nexus ™ Framework, 3. Retrieved from https://scrumorg-website-prod.s3.amazonaws.com/drupal/2017-07/Security%20Product%20Company%20Nexus%20Case%20Study_0.pdf	1	1	1	1	**4**
#25	Scrum.org. (2018). How Net Health Scales Scrum with the Nexus Framework. 3. Retrieved from https://scrumorg-website-prod.s3.amazonaws.com/drupal/2018-02/Case-Study_NetHealth_February2018_web.pdf	1	1	1	1	**4**

Source Identifier	Article Reference	Q1	Q2	Q3	Q4	Quality Score
#26	Scrum.org. (2018a). Cathay Pacific Airways Makes Nexus Their Official Scaling Framework For IT: Increases Product Delivery by 200%, 3. Retrieved from https://scrumorg-website-prod.s3.amazonaws.com/drupal/2018-09/Cathay-Pacific-Airways_Sept2018.pdf	1	1	1	1	**4**
#27	Coleman, J. (2018). How do Nexus and LeSS Differ? Retrieved from https://www.scrum.org/resources/blog/how-do-nexus-and-less-differ	1	1	1	1	**4**
#28	Verma, R. (2017). I'm Not Calling Your Baby Ugly - Two Ways and 25 Dimensions to Compare Agile Scaling Frameworks. Retrieved from https://www.scrum.org/resources/blog/im-not-calling-your-baby-ugly-two-ways-and-25-dimensions-compare-agile-scaling	1	1	1	1	**4**
#29	Digital.ai (2021). 15th State of Agile Report. Retrieved from https://digital.ai/resource-center/analyst-reports/state-of-agile-report	1	1	1	0	3
#30	van Wessel, R. et al. (2021), Scaling Agile Company-Wide: The Organizational Challenge of Combining Agile-Scaling Frameworks and Enterprise Architecture in Service Companies. Retrieved from https://ieeexplore-ieee-org.uplib.idm.oclc.org/document/9651540	1	1	1	1	4
#31	Uludağ, Ö. et al. (2021), Evolution of the Agile Scaling Frameworks. Paper presented at the Proceedings of the Agile Processes in Software Engineering and Extreme Programming.	1	1	1	1	4

Source Identifier	Article Reference	Q1	Q2	Q3	Q4	Quality Score
#32	Moe, N. et al. (2021), Finding the sweet spot for organizational control and team autonomy in large-scale agile software development. Empirical Software Engineering : An International Journal, 26 (5). doi: 10.1007/s10664-021-09967-3	1	1	1	1	4
#33	Edison, H., Wang, X., and Conboy, K. (2021), Comparing methods for large-scale agile software development: A systematic literature review. IEEE Transactions on Software Engineering. doi: 10.1109/TSE.2021.3069039	1	1	1	1	4

Appendix E: Extracted Data Form

Source Identifier	Author(s)	Title	Year of Publication	Publication Type	Peer-Reviewed	Topic	Extracted Themes
#1	Denning, S	Agile: it's time to put it to use to manage business complexity. Strategy & Leadership	2015	Journal Article	Yes	NSA	AgileOriginal, OriginalFW, ScaleNeed, ScaledPros&Con, ScaledExamples
#2	Denning, S	The age of Agile. Strategy & Leadership	2017	Journal Article	Yes	NSA	AgileOriginal, ScaledFactors, ScaledPros&Con, ScaledExamples
#3	Ebert, C., & Paasivaara, M	Scaling Agile	2017	Journal Article	Yes	SA	AgileOriginal, ScaleNeed, ScaledFW, ScaledFactors, ScaledPros&Con, ScaledExamples

#4	Cohn, M	Succeeding with Agile: Software Development Using Scrum	2010	Book		SA	OriginalFW, ScaleNeed, ScaledFW, ScaledFactors, ScaledPros&Con, ScaledExamples
#5	Reifer, D. J., Maurer, F., & Erdogmus, H.	Scaling agile methods.	2003	Journal Article	Yes	SA	AgileOriginal, OriginalFW, ScaleNeed, ScaledFactors, ScaledPros&Con
#6	Prikladnicki, R., Lassenius, C., Tian, E., & Carver, J.C.	Trends in Agile: Perspectives from the Practitioners.	2016	Journal Article	Yes	NSA	OriginalFW, ScaleNeed, ScaledFW
#7	Paasivaara, M., & Lassenius, C.	Deepening Our Understanding of Communities of Practice in Large-Scale Agile Development.	2014	Conference Paper	Yes	SA	AgileOriginal, ScaleNeed, ScaledFactors ScaledPros&Con

#8	Dikert, K., Paasivaara, M., & Lassenius, C.	Challenges and success factors for large-scale agile transformations: A systematic literature review.	2016	Journal Article	Yes	SA	AgileOriginal, OriginalFW, ScaleNeed, ScaledFW, ScaledFactors
#9	Bass, J. M.	Artefacts and agile method tailoring in large-scale offshore software development programmes.	2016	Conference Paper	Yes	SA	AgileOriginal, OriginalFW, ScaleNeed, ScaledFW, ScaledFactors, ScaledPros&Con, ScaledExamples
#10	Fourie, L., & De Vries, M.	Exploring enhancements to the agile approach for mid-sized enterprises in the services sector.	2017	Journal Article	Yes	SA	AgileOriginal, OriginalFW, ScaledFactors
#11	Conboy, K., & Carroll, N.	Implementing Large-Scale Agile Frameworks: Challenges and Recommendations.	2019	Journal Article	Yes	SA	ScaledFW, ScaledFactors, ScaledPros&Con, ScaledExamples

#12	Schwaber, K.	Nexus™ Guide The Definitive Guide to scaling Scrum with Nexus: The Rules of the Game	2018	Electronic Book	Yes	SA	OriginalFW, ScaleNeed, ScaledFW, ScaledFactors
#13	Agile, S.	SAFe® 4.6 Introduction Overview of the Scaled Agile Framework® for Lean Enterprises.	2018	White Paper	No	SA	ScaledFW, ScaledPros&Con
#14	M. Paasivaara et al.	Integrating Global Sites into the Agile and Lean Transformation at Ericsson	2013	Conference Paper	No	SA	AgileOriginal, OriginalFW, ScaleNeed, ScaledFW, ScaledFactors, ScaledPros&Con, ScaledExamples
#15	M. Paasivaara	Adopting SAFe to Scale Agile in a Globally Distributed Organization	2017	Conference Paper	No	SA	AgileOriginal, ScaleNeed, ScaledFW, ScaledFactors, ScaledPros&Con, ScaledExamples
#16	CollabNet, & VersionOne	The 12th State of Agile Report.	2018	Report	No	NSA	OriginalFW, ScaledFW,

							ScaledFactors, ScaledPros&Con
#17	M. Kalenda	Scaling Agile Software Development in Large Organizations	2017	Thesis	No	SA	AgileOriginal, OriginalFW, ScaledFW ScaledFactors, ScaledExamples
#18	Paasivaara, M., Lassenius, C., & Heikkilä, V. T	Inter-team coordination in large-scale globally distributed scrum: Do Scrum-of-Scrums really work?	2012	Conference Paper	No	SA	AgileOriginal, OriginalFW,ScaleNeed, ScaledFW, ScaledFactors, ScaledPros&Con, ScaledExamples
#19	Dolman, R., & Spearman, S.	ASK: AGILE SCALING KNOWLEDGE - THE MATRIX	2017	Web Page	No	SA	ScaledFW, ScaledPros&Con
#20	B.V., T. L. C	Introduction to LeSS	2019	Web Page	No	SA	AgileOriginal, OriginalFW, ScaleNeed, ScaledFW, ScaledExamples

#21	Agile, D	Introduction to Disciplined Agile Delivery (DAD).	n.d.	Web Page	No	SA	OriginalFW, ScaleNeed, ScaledFW, ScaledFactors, ScaledPros&Con
#22	Francino, Y.	Large-scale agile frameworks compared: SAFe vs DAD.	n.d.	Web Page	No	SA	OriginalFW, ScaleNeed, ScaledFW, SA 3
#23	QASymphony	The Pros and Cons of the Scaled Agile Framework (SAFe)	2018	Web Page	No	SA	ScaledFW, ScaledFactors
#24	Scrum.org.	Security Software Product Company Uses Nexus ™ Framework	2017	Web Page (Case Study)	No	SA	OriginalFW, ScaleNeed, ScaledFW, ScaledPros&Con, ScaledExamples
#25	Scrum.org.	How Net Health Scales Scrum with the Nexus Framework.	2018	Web Page (Case Study)	No	SA	OriginalFW, ScaleNeed, ScaledFW, ScaledPros&Con, ScaledExamples

#26	Scrum.org.	Cathay Pacific Airways Makes Nexus Their Official Scaling Framework For IT: Increases Product Delivery by 200%	2018	Web Page (Case Study)	No	SA	OriginalFW, ScaleNeed, ScaledFW, ScaledPros&Con, ScaledExamples
#27	Coleman, J.	How do Nexus and LeSS Differ?	2018	Blog	No	SA	ScaledFW, ScaledPros&Con
#28	Verma, R	I'm Not Calling Your Baby Ugly - Two Ways and 25 Dimensions to Compare Agile Scaling Frameworks.	2017	Blog	No	SA	ScaledFW, ScaledFactors
#29	Digital.ai	15th State of Agile Report	2021	Report	No	NSA	OriginalFW, ScaleNeed, ScaledFW,
#30	van Wessel, R., Kroon, P., & de Vries, H.	Scaling Agile Company-Wide: The Organizational Challenge of Combining Agile-Scaling Frameworks and Enterprise Architecture in Service Companies	2021	Journal Article	Yes	SA	AgileOriginal, OriginalFW, ScaleNeed, ScaledFW, ScaledFactors, ScaledPros&Con, ScaledExamples
#31	Uludağ, Ö., Putta, A., Paasivaara, M., & Matthes, F.	Evolution of the Agile Scaling Frameworks	2021	Conference Paper	Yes	SA	AgileOriginal, OriginalFW, ScaleNeed, ScaledFW, ScaledFactors, ScaledPros&Con

#32	Moe, N., Šmite, D., Paasivaara, M., & Lassenius, C.	Finding the sweet spot for organizational control and team autonomy in large-scale agile software development	2021	Journal Article	Yes	SA	ScaledFW, ScaledFactors, ScaledExamples
#33	Edison, H., Wang, X., & Conboy, K.	Comparing methods for large-scale agile software development: A systematic literature review	2021	Journal Article	Yes	SA	OriginalFW, ScaleNeed, ScaledFW, ScaledFactors, ScaledPros&Con

Appendix F: Data Coding (Master Set)

Appendix F is the 'Master Set' of the inter-coder agreement with the markings of what was agreed and disagreed with on the secondary set of coding. The quotes marked with a tick are those agreed with, the quotes with a crosses are those that were disagreed with, The quotes with crosses and black text are the quotes which the master set had but the secondary set did not, and the quotes with a cross and in red text are those that the master set did not have but the secondary set did have.

Source #14		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	☑ "Agile methods were originally developed for small teams."
	OriginFW	☑ "Actually, many founding ideas of this transformation came from lean, such as continuous improvement, removing waste, flow optimization in the form of end-to-end development and value stream thinking."
	ScaleNeed	☑ "During recent years large organizations have increasingly adopted agile methods, as they want to reap their claimed benefits, e.g., improved performance and fast response time. Thus, it is necessary to scale the agile practices."
Scaled Agile	ScaledFW	☑ "…the evolution of the Product Owner role and Scrum-of-Scrum meetings."
	ScaledFactors	☑ "Scaling involves challenges, such as coordination between the teams, lack of architecture, lack of requirement analysis, as well as all the challenges of distributed projects" ☑ "Besides challenges in scaling the agile practices to large projects, the transformation itself presents a formidable challenge. Transforming a large organization from a plan-driven process model to agile development is not straight forward."

		☑ "However, agile methods were originally developed for small organizations and their large-scale application has proved challenging" ☑ "Challenges relate partly to organizational size bringing inertia, which slows down the change process. Another challenge is the need to interface with and integrate existing processes and organizational structures." ☒ "Agile methods focus largely on intra-team practices, which work well in small organizations. A challenge in large organizations is that it is necessary to coordinate and communicate between several development teams, and also between different organizational units." ☑ "Because of this, large organizations must tailor and augment the methods to fit their specific needs. As a consequence, practices requiring additional formal communication may need to be institutionalized, reducing agility."
	ScaledPros&Cons	☑ "In the beginning the teams faced a lot of problems mainly due to two main impediments: firstly, the teams did not have all the needed competences, and secondly, the continuous integration environment was still in its early phases." ☒ "Second, the teams did not have enough system level design knowledge, as technical experts had previously done that and now it was expected that the teams would take care of the system design."
	ScaledExamples	☒ "This paper is based on a case study of an Ericsson node development unit distributed to three global sites: Finland, Hungary and the US. This globally distributed

		unit develops a large and complex systems product, a node that handles specific type of traffic in telecommunications networks." ☑ "In this paper we described how one R&D unit of Ericsson successfully integrated three global sites in their lean and agile transformation…"

Source #18		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	☑ "Agile methods were originally designed for small collocated teams."
	OriginFW	☑ "In this paper we focus on scaling practices for Scrum, which is one of the most widely adopted agile methods" ☑ "However, in Scrum everybody should care about the whole product, and according to Scrum, teams are responsible for inter-team coordination."
	ScaleNeed	☑ "Due to their popularity, they are nowadays also applied by large companies in large software development projects employing multiple teams that are distributed to several geographical locations."
Scaled Agile	ScaledFW	☑ "Scrum-of-Scrums meeting is mentioned in the literature as the mechanism for handling inter-team coordination in large-scale Scrum." ☑ "The Scrum-of-Scrums meeting (SoS) is basically the only practice Scrum offers for inter-team coordination" ☑ "The basic format of the SoS resembles the Daily Scrum meeting, except that it deals with teams instead

		of team members. It is recommended that the SoS be arranged daily, or 2–3 times a week." ☑ "For scaling up the SoS meetings when having a lot of teams, nested Scrum meetings, i.e., Scrum-of-Scrum-of-Scrum meetings (SoSoS) have been suggested"
	ScaledFactors	☑ "Scaling agile methods to this new context introduces new challenges, such as inter-team coordination, distribution of work without a defined architecture or properly defined requirements, as well as all the challenges of distributed projects"
	ScaledPros&Cons	☑ "The biggest challenge was the information flow up and down the nested structure. Another case study reports the usage of weekly two-level nested SoS meetings in a project consisting of 21 teams, and mention the weekly SoS as the best way to mitigate the problem of having two teams solving the same problem that a third team had already solved" ☑ "However, these meetings were not considered to work properly, and were replaced by a structure consisting of two separate meetings: a Finnish SoS followed by a Global SoS, both led by the Finnish project manager." ☑ "Initially, the representative of each team answered the four SoS questions on the behalf of his or her team. However, when the number of teams grew larger, the meetings took longer and longer and the teams gave feedback in the retrospective that they were not that interested in what the other teams were doing" ☑ "In Case A, managers admitted that inter-team communication and the SoS meetings did not work properly, and that they did not know how to improve it. The majority of the interviewed team members saw

		the SoS meetings as poor or even useless. Despite this fact, when we recently visited the company again, there was still no improvement to the situation." ☑ "The problem was that the teams did not seem to know what to share, thus often ending up just reporting "Nothing to share". As a result, participants did not find these meetings useful, and some of our interviewees even claimed that it was a waste of time." ☒ "Participants of the Grande SoS meetings did not clearly know what to share with each other, they were not interested in what others were doing and sometimes could not even understand each other's problems, since technologies used in different parts of the product differed." ☑ "SoS meetings seem to work poorly when they have too many participants with disjoint interests and concerns; and smaller, focussed interteam meetings with participants having joint goals and interests, seem to have a better chance of being perceived as successful."
	ScaledExamples	☑ "Case project A, developing a telecom product from scratch using Scrum from the outset, had grown from 2 teams to 20 teams distributed to four sites in 2,5 years. Teams were site-specific." ☑ "Case project B consisted of 25 site-specific Scrum teams distributed to two sites, developing a 10-year old telecommunications product. The organization had started a transformation from waterfall to Scrum 1,5 years ago, and at the time of the interviews all software development teams were using Scrum."

Source #26		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	
	OriginFW	☑ "The Cathay Pacific IT team is divided into IT Solutions and IT Infrastructure & Operations. They were primarily utilizing a waterfall approach with some elements of Scrum." ☒ "Scrum has worked well for us, so there was no need to adapt to a new agile framework or new mindset - doing so would have added complexity to the Scrum Teams, who were already working on complex products."
	ScaleNeed	☑ "At that time, they were experiencing a number of challenges with their software development projects, including long lead times for delivery which had the potential to threaten deadlines. They were only able to release integrated increments once every three months, which was not frequent enough to reach their desired targets." ☑ "Additionally, there were difficulties in working with some external vendors, meaning a lot of re-work was required on specific projects." ☒ "…the Product Owner did not have adequate visibility, and therefore lacked understanding of the delivery progress and what was being developed."
Scaled Agile	ScaledFW	☒ "They brought their UX and UI capability inhouse alongside their front-end and middleware development and formed three new Scrum Teams in the Nexus. They have a separate Scrum Team in the Nexus for

		backend development, which is still an external vendor, but their team lead is now located at their Hong Kong headquarters. There is one Scrum Master for each Scrum Team and a single Product Owner for IBE." ☑ "Nexus is lightweight and simple compared to other scaled Scrum frameworks in the market."
	ScaledFactors	
	ScaledPros&Cons	☒ "Nexus is lightweight and simple compared to other scaled Scrum frameworks in the market. Scrum has worked well for us, so there was no need to adapt to a new agile framework or new mindset – doing so would have added complexity to the Scrum Teams, who were already working on complex products. Nexus was the perfect fit for our organization" ☑ "With the increase in Product Backlog transparency, it made it easier for the Product Owner to order Product Backlog Items for IBE and other new projects." ☑ "Development Teams became fully aware of the dependencies and began to work more closely with other teams." ☑ "Within two months of implementing Nexus for the IBE project, the Scrum Teams increased the frequency of when they were able to release integrated increments by 200%; from one release every three months to at least once per month, and up to two to three times per month, with an improvement in quality as well."

		☑ "Within the Nexus framework, a Nexus Integration Team (NIT) is accountable for ensuring integration across teams during the Sprint, which drove focus toward integration issues and defects to be identified and fixed earlier within Sprint." ☑ "Another positive development from Nexus was the increased insight of the Product Owner. At first, the Product Owner did not have adequate visibility, and therefore lacked understanding of the delivery progress and what was being developed." ☑ "The Product Owner is now able to express what they are accomplishing and delivering in terms of value."
	ScaledExamples	☑ "In February 2017, Cathay Pacific adopted the Nexus framework developed by Scrum.org and Scrum co-creator, Ken Schwaber" ☒ "Nexus is now the official scaling framework for Cathay Pacific."

Appendix G: Data Coding (Secondary Set)

Source #14		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	"…agile methods were originally developed for small organizations and their large-scale application has proved challenging"
	OriginFW	"Actually, many founding ideas of this transformation came from lean, such as continuous improvement, removing waste, flow optimization in the form of end-to-end development and value stream thinking."
	ScaleNeed	"Agile methods were originally developed for small teams. During recent years large organizations have increasingly adopted agile methods, as they want to reap their claimed benefits, e.g., improved performance and fast response time. Thus, it is necessary to scale the agile practices."
Scaled Agile	ScaledFW	"…the goal of which was both to follow how the transformation was

		progressing and to focus on specific topics, the evolution of the Product Owner role and Scrum-of-Scrum meetings.”
	ScaledFactors	“Scaling involves challenges, such as coordination between the teams, lack of architecture, lack of requirement analysis, as well as all the challenges of distributed projects.” “Transforming a large organization from a plan-driven process model to agile development is not straight forward.” “…agile methods were originally developed for small organizations and their large-scale application has proved challenging” “Challenges relate partly to organizational size bringing inertia, which slows down the change process. Another challenge is the need to interface with and integrate existing processes and organizational structures.” “Agile methods provide little guidance on how project teams should interact with

		the environment at large. Because of this, large organizations must tailor and augment the methods to fit their specific needs."
	ScaledPros&Cons	"In the beginning the teams faced a lot of problems mainly due to two main impediments: firstly, the teams did not have all the needed competences, and secondly, the continuous integration environment was still in its early phases."
	ScaledExamples	"In this paper we described how one R&D unit of Ericsson successfully integrated three global sites in their lean and agile transformation…"

Source #18		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	"Agile methods were originally designed for small collocated teams."
	OriginFW	"…in Scrum everybody should care about the whole product, and according to

		Scrum, teams are responsible for inter-team coordination." "In this paper we focus on scaling practices for Scrum, which is one of the most widely adopted agile methods"
	ScaleNeed	"Due to their popularity, they are nowadays also applied by large companies in large software development projects employing multiple teams that are distributed to several geographical locations."
Scaled Agile	ScaledFW	"Scrum-of-Scrums meeting is mentioned in the literature as the mechanism for handling inter-team coordination in large-scale Scrum." "The Scrum-of-Scrums meeting (SoS) is basically the only practice Scrum offers for inter-team coordination" "The basic format of the SoS resembles the Daily Scrum meeting, except that it deals with teams instead of team members. It is recommended that the SoS be arranged daily, or 2–3 times a week."

		"For scaling up the SoS meetings when having a lot of teams, nested Scrum meetings, i.e., Scrum-of-Scrum-of-Scrum meetings (SoSoS) have been suggested"
	ScaledFactors	"Scaling agile methods to this new context introduces new challenges, such as inter-team coordination, distribution of work without a defined architecture or properly defined requirements, as well as all the challenges of distributed projects."
	ScaledPros&Cons	"The biggest challenge was the information flow up and down the nested structure." "… these meetings were not considered to work properly, and were replaced by a structure consisting of two separate meetings: a Finnish SoS followed by a Global SoS..." "Initially, the representative of each team answered the four SoS questions on the behalf of his or her team. However, when the number of teams grew larger, the meetings took longer and

		longer, and the teams gave feedback in the retrospective that they were not that interested in what the other teams were doing." In Case A, managers admitted that inter-team communication and the SoS meetings did not work properly, and that they did not know how to improve it. The majority of the interviewed team members saw the SoS meetings as poor or even useless." "In this project, SoS meetings were initially held using videoconferencing three times a week. Each team send a representative of their choice, most teams opting for a round-robin model. In this meeting, each team could report what they found important to share with the other teams. The problem was that the teams did not seem to know what to share, thus often ending up just reporting "Nothing to share". As a result, participants did not find these meetings useful, and some of our interviewees

		even claimed that it was a waste of time." "SoS meetings seem to work poorly when they have too many participants with disjoint interests and concerns; and smaller, focussed inter-team meetings with participants having joint goals and interests, seem to have a better chance of being perceived as successful."
	ScaledExamples	"Case project A, developing a telecom product from scratch using Scrum from the outset, had grown from 2 teams to 20 teams distributed to four sites in 2,5 years." "Case project B consisted of 25 site-specific Scrum teams distributed to two sites, developing a 10-year old telecommunications product."

Source #26		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	-
	OriginFW	"The Cathay Pacific IT team is divided into IT Solutions and IT Infrastructure &

		Operations. They were primarily utilizing a waterfall approach with some elements of Scrum." "Scrum has worked well for us, so there was no need to adapt to a new agile framework or new mindset - doing so would have added complexity to the Scrum Teams, who were already working on complex products."
	ScaleNeed	"At that time, they were experiencing a number of challenges with their doftware development projects, including long lead times for delivery which had the potential to threaten deadlines. They were only able to release integrated increments once every three months, which was not frequent enough to reach their desired targets." "…there were difficulties in working with some external vendors, meaning a lot of re-work was required on specific projects." "…the Product Owner did not have adequate visibility, and therefore lacked

		understanding of the delivery progress and what was being developed."
Scaled Agile	ScaledFW	"Nexus is lightweight and simple compared to other scaled Scrum frameworks in the market."
	ScaledFactors	
	ScaledPros&Cons	"With the increase in Product Backlog transparency, it made it easier for the Product Owner to order Product Backlog Items for IBE and other new projects" "The Development Teams became fully aware of the dependencies and began to work more closely with other teams." "Within two months of implementing Nexus for the IBE project, the Scrum Teams increased the frequency of when they were able to release integrated increments by 200%." "Within the Nexus framework, a Nexus Integration Team (NIT) is accountable for ensuring integration across teams during the Sprint, which drove focus toward

		integration issues and defects to be identified and fixed earlier within Sprint." "Another positive development from Nexus was the increased insight of the Product Owner." "The Product Owner is now able to express what they are accomplishing and delivering in terms of value."
	ScaledExamples	"In February 2017, Cathay Pacific adopted the Nexus framework…"

Appendix H: Intercoder-Agreement Comparison

Code	Agree	Disagree	Code Total	Agreement
AgileOriginal	2	0	2	100%
OriginFW	4	1	5	80%
ScaleNeed	4	1	5	80%
ScaledFW	6	1	7	86%
ScaledFactors	6	1	7	86%
ScaledPros&Cons	13	3	16	81%
ScaledExamples	4	1	5	80%

Appendix I: Thematic Analysis Coding

Source: #1		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	"In the early 2000s, the founders of agile and Scrum abandoned the romantic language of teams. Instead, they chose a language that was deliberately mundane and even ugly: Scrum, Scrum Masters, Product Owners, burndown charts, working software, sprints, standups and getting work "done."" "In retrospect, it's easy to see why agile and Scrum – which had their origins in manufacturing in Toyota several decades earlier – took off in software."
	OriginFW	"A variety of radical management methodologies for managing software development have emerged, including Scrum, Kanban, XP and Continuous Deployment. They share a common ideology sometimes called agile" "If you extract the practices of agile/Scrum from the esoteric vocabulary in which it is expressed for software developers ("sprints," "burndown charts," "product owner," "Scrum Master") the core process is:"
	ScaleNeed	"In the economy of the 20th Century, most highly successful firms were vertically integrated enterprises run by a hierarchy of executives and managers following a model of bureaucracy and an ideology of top-down control. With the advent of the Creative Economy of the 21st Century, an alternative success model is emerging: a firm that is integrated horizontally, and operating at scale in a peer-to-peer mode, with an ideology of enablement." "One reason why even successful teams were disbanded was the article of faith among traditional managers that collaborative work arrangements didn't scale. They saw an inexorable choice to be made between collaborative arrangements that could handle innovation but

		couldn't scale and bureaucratic arrangements that were efficient and callable and copied or bought innovation when it was required" ""What we need," as Hamel pointed out, "is a cluster of radically new management principles and processes that will help us take advantage of scale without becoming sclerotic, that will maximize efficiency without suffocating innovation, that will boost discipline without extinguishing freedom.""
Scaled agile	ScaledFW	
	ScaledFactors	
	ScaledPros&Cons	"Equally, some of the current efforts to "scale agile," such as the Scaled Agile Framework (SAFe), seem to be shoehorning the horizontal ideology of agile and Scrum into vertical management structures. In the process of striving for "alignment," they run the risk that the firm will emerge back in the unproductive vertical world of hierarchical bureaucracy."
	ScaledExamples	"Agile/Scrum is enjoying added attention from managers in collaborative work arrangements such as networks and ecosystems. These arrangements can be scaled without sclerosis and seemingly without limit. Apple, Autodesk and Facebook showed that it was possible to scale ecosystems of hundreds of thousands of designers and even billions of users, with relatively modest central staffs, thus combining creativity, innovation and the disciplined execution of agile/Scrum at scale"

Source: #2		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	"A fifth possibility was sketched by Julian Birkinshaw, Professor of Strategy and Entrepreneurship at the London Business School."
	OriginFW	

	ScaleNeed	
Scaled Agile	ScaledFW	
	ScaledFactors	"In short, we need to move from scalable efficiency to scalable learning, where everyone is driven by the need to learn faster and accelerate performance improvement."
	ScaledPros&Cons	"The result is faster development that is more relevant to the specific needs of the customers. The client gets value sooner. Ericsson has less work in progress. And Ericsson is deploying systems one to two years earlier than it otherwise would, so that its revenue comes in one to two years earlier." "It is now getting feedback from an active user group of more than 7 million users and is issuing updates weekly – a game changing acceleration."
	ScaledExamples	"In another session on "the entrepreneurial organization at scale" Barclays, Ericsson, Microsoft and others were cited as examples of agile" "Now with agile management, Ericsson has over 100 small teams working with its customers' needs in three-week cycles." "In 2015, Barclays announced that embracing agile was a key strategic initiative and encouraged hundreds of teams to become champions of an agile transformation." "Other parts of Microsoft such as the Developer Division and Skype are also implementing agile."

Source #3		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	"The initial agile Manifesto, based on the experiences of Microsoft, IBM, and others, primarily collected principles and practices. The label

		"agile" was a wonderful marketing stimulus and immediately triggered hype, specifically regarding small teams and low-risk products."
	OriginFW	
	ScaleNeed	"Industry soon realized that critical systems need more than an agile manifesto. Industry-scale software development typically doesn't fit with the heavy constraints of early agile practices, such as Extreme Programming. Projects easily take several years and span teams worldwide. Global software engineering demands that agile practices be scalable." "Again, these industry developments require agile practices to be scalable." "Agility has arrived in real-world development, beyond mere software applications. Development of even critical systems will evolve to a continuous process that fully decouples the rather stable hardware from delivered services driven by continuous software upgrades." "Barriers to using agile technologies no longer exist. Developments in globally distributed teams, large projects, safety-critical systems, and hardware and systems engineering have showed that agile technologies are adoptable and adaptable."
Scaled Agile	ScaledFW	"Current approaches for scaling agile blend agile and lean practices to address real industry needs." "For example, in a recent survey, almost 30 percent of the respondents said they used the Scaled Agile Framework (SAFe) to provide scaled agility" "Many practitioners consider SAFe too heavy and complex." "Some even say that SAFe adds to bureaucracy, evolving to "the new waterfall."" "Other agile frameworks, such as Large-Scale Scrum (LeSS) and Lean Scalable Agility for Engineering (LeanSAFE), have addressed this high complexity. They define much less and give more freedom to tailoring."

	ScaledFactors	"Both companies customized the framework to suit their organization and found that to be one of the success factors. Other success factors were investing in the first program increment planning event to make it a success, employing external consultants to train and coach, having active internal change agents, having strong leadership support, and quickly reacting to feedback to continuously improve and remove the adoption's pain points." "The most serious challenges included change resistance, lack of communication of the adoption, lack of continuous improvement, and lack of training and support during the adoption." "Because constraints vary across application domains, such as liability and governance, agile practices must be tailored to needs and risks. To address such tailoring, companies can use agile scaling frameworks, which can be applied across the organization." "From working with many companies worldwide on implementing these frameworks, we've found that introducing agile development means changing the culture and mind-set. It requires long-term commitment, big investments, and customization to a company's specific situation." "The biggest, most difficult change is that of the mind-set." "Changing the practices won't make a company agile if the underlying culture and thinking don't change."
	ScaledPros&Cons	"Managers find it comfortable because it has plenty of role definitions, which hasn't been explicit in the classic agile toolkits used in the past two decades. On the other hand, many users perceive such a highly prescriptive role-and-process scheme as overhead and not any more agile." "More frequent and more predictable releases with better quality, improved visibility and communication across the globally distributed organization, must be designed and achieved on the systems-engineering level. Enhance reuse across platforms, products, and markets. Evolve to self-x-type architectures and technologies such as self-aware adaptive systems to cope with fast-changing components and environments."

	ScaledExamples	"We compared the adoption of the Scaled Agile Framework (SAFe) in two business lines of Comptel, a globally distributed organization that was recently acquired by Nokia."

Source #4		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	
	OriginFW	"The teams had become accustomed to the iterative and incremental nature of Scrum…"
	ScaleNeed	"Large Projects are often more critical to the organisation, under greater scrutiny, and more time-sensitive, more prone to personality clashes, longer, and more likely to be distributed across multiple sites" "On a large project with multiple teams, however, the product owner role is too big for one person, so we must find ways of scaling it."
Scaled Agile	ScaledFW	"A fairly universal practice for coordinating work among several teams is the scrum of scrums meeting. These meetings allowed clustered to of teams to discuss their work, focusing especially on areas of overlap and integration" "If a large project is being built by many team of teams, one representative of each scrum of scrums can attend what might be called "scrum of scrum of scrums"… most organisations stick to calling it a scrum of scrums meeting regardless of the level at which it is occurring."
	ScaledFactors	"If there's only one product, there should be only one product backlog." "As a project grows to include multiple teams, ideally a new product owner is count for each"

		"Fortunately, there are additional techniques Scrum teams can employ to further manage dependencies. These include doing rolling lookahead planning, holding kick-off meetings, sharing team members, and even using a dedicated integration team." "An integration team works in the gaps that may exist between development teams. Most of these gaps occur in the interfaces between teams. Interface problems can be broadly split into these two categories: Unidentified interfaces … unattended interfaces." "This conservatism wasn't because agile of Scrum turned out to be unsuitable for large project but because they hadn't used these processes on large projects and were reluctant to advise readers to go so. But, in the years since the agile Manifesto and the books that came shortly before and after it, we have learned that the principles and practices of agile development can be scaled up and applied on large projects, albeit it with a considerable amount of overhead ."
	ScaledPros&Cons	"… I hear the of "our product backlog is too big," it is almost always in reference to a product backlog with 100 or more items." "Because Scrum scales by having multiple small teams rather than one large team, the problem arises of how to coordinate the work of all those teams."
	ScaledExamples	"As an example, consider a large San Fransisco-based bioinformatics company that has two dozen feature teams, and one integration team."

Source #5		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	"A sweet-spot agile project typically involves a small, self-organizing, collocated team of fewer than 20 developers and one or more on-site customers. The team usually works together on a variable-scope application with unstable or emergent requirements, relying mainly on an oral culture based on high-bandwidth, face-to-face communication."

	OriginFW	"The industrial delegates, who had been putting agile methods to work (using lightweight methods like Crystal, Extreme Programming, Dynamic Systems Development Method, Feature-Driven Development, Scrum, and a scaled-down version of the Rational Unified Process)"
	ScaleNeed	"If large teams are to produce lots of software functionality quickly, the agile methods involved must scale to meet the task." "Scaling agile teams thus becomes an issue if the only option for meeting a system delivery deadline is to have many developers working concurrently."
Scaled Agile	ScaledFW	
	ScaledFactors	"In a departure from current thinking, delegates working on large projects agreed that some architectural development was needed before pushing ahead with iterations." "For example, some suggested that the concept of a daily team meeting could extend to include a daily intrateam project meeting. Others discussed the issues associated with scaling planning meetings and requirements engineering concepts." "Use an integration team …" "Most delegates agreed that agile methods fit small projects where problems of scale are minor. However, difficulty arises in scaling these methods to fit large projects where teams of teams work together." "Because Scrum scales by having multiple small teams rather than one large team, The problem arises of how to coordinate the work of all those team."
	ScaledPros&Cons	"For large projects, participants advocated the benefits of using daily project meetings where team leaders get together to work through issues; identify desired functionality; and coordinate release contents, iterations, and progress."

		"Larger teams tend to be less tolerant to change. Adopters of agile methods should therefore be extremely conservative when setting expectations for change, especially when using these new methods on large projects."
	ScaledExamples	

Source #6		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	
	OriginFW	"Noting that Scrum forms the basis of most agile implementations in industry …"
	ScaleNeed	"globally dispersed teams that can collaborate and function just as well as collocated teams"
Scaled Agile	ScaledFW	"These include the addition of values to Scrum, the Nexus framework for scaling Scrum up to nine teams working on the same product, and a set of metrics for evaluating Scrum implementations."
	ScaledFactors	
	ScaledPros&Cons	
	ScaledExamples	

Source #7		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	"Agile software development methods were originally designed for single teams of less than 10 people."
	OriginFW	

	ScaleNeed	"However, during recent years, large organizations have increasingly adopted agile software development, leading to a need for scaling the methods"
Scaled Agile	ScaledFW	
	ScaledFactors	"Scaling involves challenges such as coordination between several agile teams, lack of architectural planning, lack of requirement analysis, as well as all the challenges of distributed projects, as many large organizations are distributed"
	ScaledPros&Cons	"Scrum-of-Scrum meetings of representatives from around 25 teams were tried for a while, but that did not work, as these teams did not share enough common interests." "Instead, so called Feature Coordination CoPs, a kind of Scrum-of-Scrum meetings for a few teams working on a common feature, emerged as a well-working mechanism."
	ScaledExamples	

Source #8		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	"Agile methods were originally designed for use in small, single- team projects"
	OriginFW	"Two of the most popular agile methods are Extreme Programming (XP) and Scrum. Scrum is a method focusing on the project management viewpoint of agile development prescribing timeboxing, continuous tracking of project progress, and customer centricity. The XP development method is a collection of practices for enabling efficient incremental development." "In early work on agile, Fowler considers the Crystal methodology to be suitable for up to 50 people"

		"The most prevalent agile method used in the transformed organizations was Scrum, which was the sole agile method mentioned in 25 cases. The second most mentioned agile method was Extreme Programming (in 4 cases), which was often combined with Scrum (in 5 cases). Lean software development was mentioned in 6 studies, although in all cases in combination with Scrum."
	ScaleNeed	"Moreover, among the top ten items three focused on distributed agile development, which is relevant especially for larger organizations as they are often geographically distributed"
Scaled Agile	ScaledFW	"Some agile methods, e.g., Scrum have suggestions: Scrum proposes the use of Scrum-of-Scrums as the main scaling practice. Several frameworks by consultants have been developed and are actively promoted, e.g. SAFe and LeSS."
	ScaledFactors	"Compared to small projects, which are ideal for agile development, larger ones are characterized by the need for additional coordination." "A particular problem in applying agile to larger projects is how to handle inter-team coordination." "The difficulty of introducing agile methods increases with the organization size" "The difficulty is partly related to size bringing higher organizational inertia which slows down organizational change" "One significant difference between small and large-scale adoptions is that larger organizations have more dependencies between projects and teams. This increases the need for formal documentation and thus reduces agility. In addition to inter-team coordination, development teams must interact with other organizational units, which are often non-agile in nature." "Size had been regarded in terms of size in persons or teams, project budget, code base size, and project duration"

		"Based on these findings, we defined large-scale to denote software development organizations with 50 or more people or at least six teams. All people do not need to be developers, but must belong to the same software development organization developing a common product or project, and thus have a need to collaborate." "Distribution had negative effects, such as missing kick-off meetings, reduced feelings of proximity when telecommunication is necessary, and difficulty in arranging frequent meetings due to time zone differences"
	ScaledPros&Cons	
	ScaledExamples	

Source #9		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	"However, such agile methods are conventionally associated with small, co-located development teams." "In small agile teams, the whole team takes on-going responsibility for developing architecture during the development project."
	OriginFW	"There are a range of agile software development methods that are increasingly being adopted in large-scale offshore software development programmes, including Feature Driven Development, Scrum, Extreme Programming (XP) and Lean Software Development" "These software development methods build upon three key themes in software engineering: development using short iterations, feature-driven development and the close interaction with customers."
	ScaleNeed	"Teams working in parallel with each other need coordination, consequently the scrum masters from each team work together to

		coordinate activities, manage dependencies and avoid the duplication of effort"
Scaled Agile	ScaledFW	"For example, the scrum of scrums approach supports multiple concurrent scrum teams." "The Scaled Agile Framework (SAFe) is targeted at large enterprises and has three main layers: team, programme and portfolio. SAFe comprises an implementation strategy, nine lean agile principles and extends the scrum of scrums concept." "The scrum of scrums meeting, in which scrum masters from co-operating teams meet to provide each other status updates, has previously been investigated" "The application of agile methods in large-scale and geographically distributed software development programmes is an area of emerging interest. Whereas agile early adopters tended to use engineering practices from the XP process, this research confirms more recent findings showing an increase in the popularity of scrum process orchestration practices"
	ScaledFactors	"So, large-scale agile development programmes dictate forms of documentation to coordinate the activities of groups and teams, yet agile methods advocate focus on working code." "However, in large-scale agile, teams cooperate to build the overall system, sometimes subject to corporate governance or third party development standards, and so cannot realistically be jointly responsible for architecture."
	ScaledPros&Cons	"It was found that, in large- scale projects, the scrum of scrums meeting membership can grow making it difficult to expedite the meeting in the recommended 15 minutes. Further, scrum of scrum meeting participants did not find it useful to listen to status updates from teams working on relatively unrelated aspects of the development programme." "Both studies found practitioners preferring to conduct a more frequent series of smaller, more focused, scrum of scrum meetings to share

		information about related aspects of the development programme, alongside a less frequent large scrum of scrums meeting involving all stakeholders." "There is a paucity of agile artefacts for scaling, the teams in this study used the scrum of scrums meetings to identify and resolve dependencies between teams, but this seemed to be a recurring challenge."
	ScaledExamples	"For example, a method engineering approach appears to have been used at Intel Shannon where aspects of both scum and XP were adopted"

Source #10		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	"Agile software development is more than a method: it is a philosophy. It is a way of thinking about software development, and is described by the four values and twelve principles that form part of the 'agile manifesto'." "They identified a need for an alternative to document-driven, heavyweight software development processes. To survive and compete in the era of the internet, companies should focus on delivering what matters to the customer in a timely, tangible and 'as promised' way"
	OriginFW	"Consequently, they decided to study the agile processes against the Agile Alliance principles and the assumptions that underpin a number of the more common agile software development methods, such as Extreme Programming and Scrum."
	ScaleNeed	
Scaled Agile	ScaledFW	
	ScaledFactors	"They concluded that agile software development methods were originally designed for small and individual project teams, and that they therefore present challenges when introducing agile software development methods at scale. They identified nine main challenges: (1)

		it is difficult to manage dependencies between agile software development teams and the broader enterprise; (2) the goals of the agile software development teams and the broader enterprise are often mis-aligned; (3) it is difficult to maintain technical consistency between different agile software development teams; (4) using traditional and new agile methods side-by-side in an enterprise creates conflict; (5) silos that exist within a company make the implementation of agile methods difficult; (6) the majority of software agile development methods lack management of high-level requirements; (7) ambiguity about requirements influences project quality from the perspective of testing functionality; (8) there is a misunderstanding of the basic agile concepts that underpin the agile manifesto when adapting agile principles; and (9) it is difficult for some teams to design for the full picture when an incremental approach is followed." "These showed that the agile software development family of methods does not naturally fit into the context of medium to large enterprises." "Prominent challenges include: agile adoption due to change management, or enterprise agility challenges, requirements management and collaboration challenges, and challenges associated with agile practices."
	ScaledPros&Cons	
	ScaledExamples	

Source #11		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	
	OriginFW	
	ScaleNeed	

Scaled Agile	ScaledFW	"To address these issues, many have turned to large-scale agile development frameworks, such as the Scaled Agile Framework (SAFe), Large-Scale Scrum (LeSS), Spotify, Nexus, and Scrum at Scale." "Some even question if certain frameworks, such as Spotify and SAFe, really have enough substance to be considered a framework or method." "Large-scale frameworks, such as SAFe, are much more dominant and small realignments can cause significant disruption across units of an organization."
	ScaledFactors	"While there are many potential benefits, large-scale transformations are fraught with challenges, such as communication issues, a lack of flexibility, and coordination challenges." "Team-level inconsistencies can be ironed out quickly, but differences across a large swath of teams "get ingrained and the differences grow and fester."""
	ScaledPros&Cons	"There were many cases where inconsistent meanings and interpretations were problematic." "As one person stated, "The very people imposing it still require the old reports and five-year plans that SAFe is supposed to eliminate." "When a formal framework, such as SAFe, Scrum at Scale, or Spotify, is used, there is a tendency to measure agile transformation by adherence to that framework, rather than the value it provides." "Also, explanations of large-scale frameworks, such as SAFe and Scrum at Scale, tend to describe their associated structures and processes but provide little guidance on how organizations can assess their overall readiness or appetite to undertake a large-scale agile transformation process." "Publications that launched frameworks, such as SAFe and Spotify, explain the basics very well, but once one applies them outside of their intended context of a specific framework, guidance quickly runs out."
	ScaledExamples	"We draw on 15 years of research collaboration experience with global companies, in particular those listed in Table 1"

Source #12		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	
	OriginFW	"Many developers have used the Scrum framework to work collectively to develop and deliver an Increment of working software."
	ScaleNeed	"However, if more than one Scrum Team is working off the same Product Backlog and in the same codebase for a product, difficulties often arise. If the developers are not on the same collocated team, how will they communicate when they are doing work that will affect each other? If they work on different teams, how will they integrate their work and test the Integrated Increment? These challenges appear when two Scrum Teams are integrating their work into a single increment, and become significantly more difficult when three or more Scrum Teams integrate their work into a single increment"
Scaled Agile	ScaledFW	"Nexus is a framework for developing and sustaining scaled product and software delivery initiatives. It uses Scrum as its building block." "Nexus is a framework consisting of roles, events, artifacts, and rules that bind and weave together the work of approximately three to nine Scrum Teams working on a single Product Backlog to build an Integrated Increment that meets a goal." "Nexus is a process framework for multiple Scrum Teams working together to create an Integrated Increment. Nexus is consistent with Scrum and its parts will be familiar to those who have used Scrum. The difference is that more attention is paid to dependencies and interoperation between Scrum Teams, delivering at least one "Done" Integrated Increment every Sprint."

		"Nexus roles, events, and artifacts inherit the purpose and intent attributes of the corresponding Scrum roles, events, and artifacts, as documented in the Scrum Guide" "A Nexus consists of a Nexus Integration Team and approximately three to nine Scrum Teams." "Just like its building block, Scrum, Nexus is based on transparency. The Nexus Integration Team works with the Scrum Teams within a Nexus and the organization to ensure that transparency is apparent across all artifacts and that the integrated state of the Integrated Increment is widely understood."
	ScaledFactors	"There are many dependencies that arise between the work of multiple teams that collaborate to create a complete and "Done" Increment at least once every Sprint. These dependencies are related to: Requirements: The scope of the requirements may overlap, and the manner in which they are implemented may also affect each other. That knowledge should be considered when ordering the Product Backlog and selecting Product Backlog items Domain knowledge: The people on the teams have knowledge of various business and computer systems. Their knowledge should be distributed across the Scrum Teams to ensure that the teams have the knowledge they need to do their work, to minimize interruptions between Scrum Teams during a Sprint. Software and test artifacts: The requirements are, or will be, instantiated in software. To the extent that requirements, team members' knowledge, and software artifacts are mapped to the same Scrum Teams, teams can reduce the number of dependencies between them. When software delivery using Scrum is scaled, these dependencies of requirements, domain knowledge, and software artifacts should drive the organization of the Development Teams. To the extent that it does, productivity will be optimized."
	ScaledPros&Cons	
	ScaledExamples	

Source #13		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	
	OriginFW	
	ScaleNeed	
Scaled Agile	ScaledFW	"More than the sum of its parts, SAFe is a scalable and configurable framework that helps organizations deliver new products, services, and solutions in the shortest sustainable lead time" "SAFe 4.6 introduces the Five Core Competencies of the Lean Enterprise—Lean-Agile Leadership, Team and Technical Agility, DevOps and Release on Demand, Business Solutions and Lean Systems Engineering, and Lean Portfolio Management." "• Essential SAFe is the basic building block for all other SAFe configurations and is the simplest starting point for implementation. It brings the core competencies of Lean-Agile Leadership, Team and Technical Agility, and DevOps and Release on Demand to the enterprise . • Large Solution SAFe brings the Business Solutions and Lean Systems Engineering competency to those building the largest and most complex solutions. This configuration supports multiple Agile Release Trains (ARTs) and suppliers. • Portfolio SAFe applies the Lean Portfolio Management competency to align portfolio execution to the enterprise strategy, and organizes development around the flow of value through one or more value streams. • Full SAFe is the most comprehensive version that integrates all five core competencies to support enterprises that build and maintain a portfolio of large, integrated solutions." "SAFe Agile teams typically: • Use a blend of Agile methods, including Scrum and Kanban

		• Base their work on short iterations, apply small user stories, plan work for the upcoming iteration, and meet daily to coordinate their work toward iteration goals • Demo the working system at the end of the iteration and retrospect on how to improve the process • Visualize and manage their flow of work in a Kanban system that helps identify bottlenecks and controls work-in-process (WIP) Agile teams work together in an organizational construct called the Agile Release Train (ART). These trains bring together all the people needed to define, build, test, and deploy (and often operate) a solution. All teams on an ART plan, integrate, demo, deploy, release, and learn together. Each team understands and commits to achieving not only their objectives but the larger ART objectives as well." "SAFe provides more than just a modular way to scale the reliable flow of software value. The Framework includes components that address the specific needs of these very large and complex cyber-physical systems, such as requirements analysis, business capability definition, functional analysis and allocation, design synthesis, verification and validation (V&V), design alternatives, trade studies, modeling, and simulation."
	ScaledFactors	
	ScaledPros&Cons	"SAFe dramatically improves business agility by accelerating productivity, time-to-market, quality, employee engagement, and more." "SAFe for Government is a set of success patterns that help public sector organizations implement Lean-Agile practices in a government context."
	ScaledExamples	

Source #14		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	"…agile methods were originally developed for small organizations and their large-scale application has proved challenging"
	OriginFW	"Actually, many founding ideas of this transformation came from Lean, such as continuous improvement, removing waste, flow optimization in the form of end-to-end development and value stream thinking."
	ScaleNeed	"During recent years large organizations have increasingly adopted agile methods, as they want to reap their claimed benefits, e.g., improved performance and fast response time. Thus, it is necessary to scale the agile practices."
Scaled Agile	ScaledFW	"…the evolution of the Product Owner role and Scrum-of-Scrum meetings."
	ScaledFactors	"Scaling involves challenges, such as coordination between the teams, lack of architecture, lack of requirement analysis, as well as all the challenges of distributed projects." "Besides challenges in scaling the agile practices to large projects, the transformation itself presents a formidable challenge. Transforming a large organization from a plan-driven process model to agile development is not straight forward." "Challenges relate partly to organizational size bringing inertia, which slows down the change process. Another challenge is the need to interface with and integrate existing processes and organizational structures." "However, agile methods were originally developed for small organizations and their large-scale application has proved challenging" "Agile methods focus largely on intra-team practices, which work well in small organizations. A challenge in large organizations is that it is

		necessary to coordinate and communicate between several development teams, and also between different organizational units." "Because of this, large organizations must tailor and augment the methods to fit their specific needs. As a consequence, practices requiring additional formal communication may need to be institutionalized, reducing agility."
	ScaledPros&Cons	"The main benefits the organization saw in agile software development was the breaking down of functional silos and forming cross-functional teams, having a close relationship with the customers, and focusing on communication rather than documentation." "Other benefits and requirements of agile development, such as quick feedback loops at several levels, and continuous integration were accomplished only later." "In the beginning the teams faced a lot of problems mainly due to two main impediments: firstly, the teams did not have all the needed competences, and secondly, the continuous integration environment was still in its early phases."
	ScaledExamples	"In this paper we described how one R&D unit of Ericsson successfully integrated three global sites in their lean and agile transformation…"

Source #15		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	"Agile methods have become commonplace in software development organizations around the world, both small and large. Originally, the methods were designed for small, collocated projects"
	OriginFW	

	ScaleNeed	"Large software development organizations adopting agile methods need solutions and models to help scale agile to fit their needs." "According to the largest reoccurring survey on agile adoption, the State of Agile Survey, 43% of the self-selected respondents worked in development organizations with more than 50% of teams using agile, and 62% of almost 4000 respondents came from an organization with over a hundred people in software development. While this survey is not scientific, it indicates that a significant number of big organizations use agile."
Scaled Agile	ScaledFW	"During recent years, several frameworks for scaling agile have been created by consultants, including the Scaled Agile Framework (SAFe), Large-scale Scrum (LeSS) and Disciplined Agile Delivery (DAD)." "According to the State of Agile Survey, the Scaled Agile Framework (SAFe) seems to be the most popular framework for scaling agile." "The Scaled Agile Framework, SAFe claims to provide arecipe for adopting agile at the enterprise scale. It contains the levels of teams, programs, and portfolio, as well as the optional value stream level." "For adopting SAFe, the SAFe 4.0 Whitepaper suggests "Implementing SAFe 1-2-3" pattern, which includes the following three steps: 1) train implementers and Lean-Agile change agents, 2) train all executives, managers, and leaders, and 3) train teams and launch agile release trains." "The business line adopting SAFe later was, based on the interviews, clearly more successful in its' transformation, which can be partially explained by learning from the experiences of the first case." "We identified the following success factors: 1) training the personnel well in advance, 2) informing and engaging people, 3) involving change agents, 4) hiring an experienced external consultant to train, advice and support, 5) preparing well for the first PI planning event, 6) having a full-time RTE and 7) taking recognized improvement items seriously by assigning responsibilities and monitoring their implementation."

	ScaledFactors	"Scaling agile is not easy, as large projects often are globally distributed, and have a large number of teams that need to collaborate and coordinate." "Comparing these findings to the success factors of large-scale agile transformations in general, the four first mentioned items are high on the literature review list. Thus, these seem to be success factors for all kinds of largescale agile transformations, not only for SAFe transformations."
	ScaledPros&Cons	"Even though distributed events are often challenging, our interviewees found these events quite successful and did not mention global distribution as a problem, but instead felt that communication channels worked well."
	ScaledExamples	"However, during recent years many large organizations have made the transition from traditional, plandriven waterfall type methods towards agile. Examples include Nokia, Ericsson, Amazon and British Telecom" "Earlier, Comptel had used a traditional, waterfall type, stage gate model in software development. In 2008 the company started to move towards agile by adopting Scrum in the software development teams. However, product management still remained in the waterfall world with 1–1,5 year plans, clear milestones and yearly software releases. The SAFe adoption followed seven years later: One of the business lines (Case 1) started the SAFe adoption in the middle of 2015 and the other business line (Case 2) followed half a year later, in the end of 2015."

Source #16		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	

	OriginFW	"Agile Methodologies Used: Scrum, ScrumBan and Scrum/XP Hybrid (70%) continue to be the most common agile methodologies used by respondents' organizations."
	ScaleNeed	
Scaled Agile	ScaledFW	"The Scaled Agile Framework® (SAFe®) is reported as the most widely-used approach to scaling agile, with nearly 1/3 (29%) saying that SAFe is the method they 'follow most closely'."
	ScaledFactors	"Respondent indicated the most valuable in helping them scale agile practices were: 1.Internal Coaches 2.Consistent practices and processes across teams. 3.Implementation of a common tool across teams. 4.External agile consultants or trainers. 5.Executive sponsorship." "The three most significant challenges to agile adoption and scaling are reported as Organizational culture at odds with agile values (53%), General organizational resistance to change (46%), and Inadequate management support and sponsorship (42%)." "Internal agile coaches (53%), consistent practices and processes across teams (43%), and the implementation of a common tool across teams (41%) are the top three factors reported to have been most helpful in scaling agile."
	ScaledPros&Cons	"By implementing agile, respondents cited seeing improvements in the following areas: Ability to manage changing priorities, Project visibility, Business/IT alignment, Delivery speed/time to market, Increased team productivity, Team morale, Project predictability, Software quality, Project risk reduction, Engineering discipline, Managing distributed teams, Software maintainability, Project cost reduction."
	ScaledExamples	

Source #17		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	"Agile methods have become a very popular approach to managing software development processes. The goal of Agile methods has been to improve management and execution of software development projects by increasing their flexibility. The focus of Agile methods was originally on small, one team projects"
	OriginFW	"It describes how Agile team works. It uses Scrum, Kanban and XP techniques in defined development process of Agile teams." "The most utilized Agile method was Scrum. Use of Scrum was mentioned in every article. The second most applied Agile method was Lean, five articles reported the use of Lean practices. Some ofthe used Lean principles were Pull Principle, Takt, Go and See and Kaizen, the most used Lean principle was Pull Principle. Two cases adopted Kanban and two cases used some framework to scale the Agile development."
	ScaleNeed	
Scaled Agile	ScaledFW	"According to the most popular are Scrum of Scrums, Scaled Agile Framework (SAFe), Large Scale Scrum (LeSS) and Disciplined Agile (DAD)." "According to the most used Agile scaling methods are Scrum/Scrum of Scrums (72 %), Scaled Agile Framework (27 %), Lean Management (17 %), Enterprise Agile (9 %), Enterprise Scrum (9 %), Agile Portfolio Management (9 %), Large Scale Scrum (6%), Disciplined Agile Delivery (4 %). Other internally created methods are used by 17 % of organizations."

		"SAFe is probably the most complex and thorough Agile scaling method. It is a complete framework, and many other mentioned practices, such as Lean Management or Agile Portfolio Management are already incorporated into it. Large Scale Scrum (LeSS) is also a full-featured framework for scaling Agile development. However, it tries to be as minimalistic as possible. It relies less on processes and specific organizational structures and more on a mindset of people and ad-hoc communication." "The framework is built upon ideas of Agile development, Lean product development, and systems thinking. SAFe deals with both software and systems development. It supports companies of different sizes, from small ones with less than 100 employees to the largest enterprises with more than thousands of people. To support such a degree of flexibility SAFe has optional extensions for large companies." "The Large Scale Scrum is another framework such as the Scaled Agile Framework. It tries to bring all necessary information about scaling in one compact solution." "Although it is still a framework, it is much more lightweight than the SAFe. It extends Scrum so it could be used in large-scale context. The core idea of LeSS is that even for large organizations there is not a need for an overcomplicated process that creates the burden of bureaucracy. LeSS tries to be compliant to one of the core principles of the Agile manifesto—that individuals and interactions are more valuable than processes and tools." "The Large Scale Scrum has two variants: LeSS and LeSS Huge. LeSS is for organizations which develop products that need up to 8 Agile teams. LeSS Huge is for enterprises that require a higher number of teams." "LeSS is intended for companies with up to eight Scrum teams developing one product." "LeSS lightly extends Scrum by adding cross-team meetings and promoting some principles. On the contrary, SaFE defines multiple levels, which each had precisely defined roles and their

		responsibilities. Agendas of meetings, organization processes, and workflow are described in detail." "We identified eight common scaling practices which the frameworks utilized: Scrums of Scrums Communities of Practice Scaled Sprint Demo Scaled Requirements Management Scaled Sprint Planning Scaled Retrospective Feature Teams Undone Department" "We identified six Agile scaling practices that the case organization used. The identified scaling practices were: Scaled Retrospective, Scaled Planning, Scrum of Scrums, Requirements Management Scaling, Undone Department and Communities of Practice."
	ScaledFactors	"Introducing Agile methods in large organizations is very challenging because the larger the organization is, the harder is to introduce changes. Scaling of Agile methods has created a lot of unique challenges, such as synchronization between teams and interaction with other organizational units." "Large projects require appropriate coordination and communication between teams, dependencies between teams need to be managed, and other non-Agile units need to be involved."
	ScaledPros&Cons	
	ScaledExamples	"Adoption of Agile methods was conducted in several large-scale projects in organizations such as Nokia, Yahoo!, Ericsson, British Telecom and Amazon."

Source #18

Topic	Theme	Quotes
Non-Scaled Agile	AgileOriginal	"Agile methods were originally designed for small collocated teams."
	OriginFW	"…in Scrum everybody should care about the whole product, and according to Scrum, teams are responsible for inter-team coordination." "In this paper we focus on scaling practices for Scrum, which is one of the most widely adopted agile methods"
	ScaleNeed	"Due to their popularity, they are nowadays also applied by large companies in large software development projects employing multiple teams that are distributed to several geographical locations."
Scaled Agile	ScaledFW	"Scrum-of-Scrums meeting is mentioned in the literature as the mechanism for handling inter-team coordination in large-scale Scrum." "The Scrum-of-Scrums meeting (SoS) is basically the only practice Scrum offers for inter-team coordination" "The basic format of the SoS resembles the Daily Scrum meeting, except that it deals with teams instead of team members. It is recommended that the SoS be arranged daily, or 2–3 times a week." "For scaling up the SoS meetings when having a lot of teams, nested Scrum meetings, i.e., Scrum-of-Scrum-of-Scrum meetings (SoSoS) have been suggested"
	ScaledFactors	"Scaling agile methods to this new context introduces new challenges, such as inter-team coordination, distribution of work without a defined architecture or properly defined requirements, as well as all the challenges of distributed projects."
	ScaledPros&Cons	"The biggest challenge was the information flow up and down the nested structure. Another case study reports the usage of weekly two-level nested SoS meetings in a project consisting of 21 teams, and mention the weekly SoS as the best way to mitigate the problem of

		having two teams solving the same problem that a third team had already solved" "However, these meetings were not considered to work properly, and were replaced by a structure consisting of two separate meetings: a Finnish SoS followed by a Global SoS, both led by the Finnish project manager." "Initially, the representative of each team answered the four SoS questions on the behalf of his or her team. However, when the number of teams grew larger, the meetings took longer and longer and the teams gave feedback in the retrospective that they were not that interested in what the other teams were doing" "In Case A, managers admitted that inter-team communication and the SoS meetings did not work properly, and that they did not know how to improve it. The majority of the interviewed team members saw the SoS meetings as poor or even useless. Despite this fact, when we recently visited the company again, there was still no improvement to the situation." "The problem was that the teams did not seem to know what to share, thus often ending up just reporting "Nothing to share". As a result, participants did not find these meetings useful, and some of our interviewees even claimed that it was a waste of time." "Participants of the Grande SoS meetings did not clearly know what to share with each other, they were not interested in what others were doing and sometimes could not even understand each other's problems, since technologies used in different parts of the product differed." "SoS meetings seem to work poorly when they have too many participants with disjoint interests and concerns; and smaller, focussed interteam meetings with participants having joint goals and interests, seem to have a better chance of being perceived as successful."
	ScaledExamples	"Case project A, developing a telecom product from scratch using Scrum from the outset, had grown from 2 teams to 20 teams distributed to four sites in 2,5 years. Teams were site-specific."

		"Case project B consisted of 25 site-specific Scrum teams distributed to two sites, developing a 10-year old telecommunications product. The organization had started a transformation from waterfall to Scrum 1,5 years ago, and at the time of the interviews all software development teams were using Scrum."

* Source #19 was not coded in the template manner as it is an Excel spreadsheet with only the scaled Agile frameworks and notes on each, therefore it speaks directly and only to the themes 'ScaledFW' and 'ScaledPros&Cons'.

Source #20		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	"In 2002, when Craig wrote Agile & Iterative Development, many believed that agile development was only for small groups."
	OriginFW	"Scrum is an empirical-process-control development framework in which a cross-functional self-managing Team develops a product in an iterative incremental manner. Each timeboxed Sprint, a potentially shippable product increment is delivered and, ideally, shipped" "A small Team is responsible for delivering the Sprint goal; there are no limiting single-specialized roles. A Scrum Master teaches why Scrum and how to derive value with it, coaches the Product Owner, Team, and organization to apply it, and acts as a mirror. There is no project manager or team lead."
	ScaleNeed	"However, we both (Craig and Bas) became interested in—and got increasing requests—to apply Scrum to large, multi-site, and offshore development"
Scaled Agile	ScaledFW	"LeSS is Scrum—Large-Scale Scrum (LeSS) isn't new and improved Scrum. And it's not Scrum at the bottom for each team, and something different layered on top. Rather, it's about figuring out how to apply the principles, purpose, elements, and elegance of Scrum in a large-scale context, as simply as possible. Like Scrum and

		other truly agile frameworks, LeSS is "barely sufficient methodology" for high-impact reasons." "In terms of large, what's a typical LeSS adoption case? Perhaps five teams in one or two sites. We've been involved in adoptions of that size, of a few hundred people, and up to a LeSS Huge case of well over a thousand people, far too many development sites, tens of millions of lines of C++, with custom hardware." "It isn't new and improved Scrum. Rather, LeSS is about figuring out how to apply the principles, rules, elements, and purpose of Scrum in a large-scale context, as simply as possible." "LeSS. 2–8 Teams LeSS Huge. 8+ Teams"
	ScaledFactors	
	ScaledPros&Cons	
	ScaledExamples	"Today, the two LeSS frameworks (smaller LeSS and LeSS Huge) have been adopted in big groups worldwide in disparate domains: telecom equipment — Ericsson & Nokia Networks investment and retail banks — UBS trading systems — ION Trading marketing platforms and brand analytics — Vendasta video conferencing — Cisco online gaming (betting) — bwin.party offshore outsourcing — Valtech India"

Source #21		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	

	OriginFW	"Many organizations start their agile journey by adopting Scrum because it describes a good strategy for leading agile software teams. However, Scrum is only part of what is required to deliver sophisticated solutions to your stakeholders. Invariably teams need to look to other methods to fill in the process gaps that Scrum purposely ignores."
	ScaleNeed	"When many people hear "scaling" they often think about large teams that may be geographically distributed in some way. This clearly happens, and people are clearly succeeding at applying agile in these sorts of situations" "Organizations are also applying agile in compliance situations, either regulatory compliance that is imposed upon them or self selected compliance (such as CMMI and ISO). They are also applying agile in a range of problem and solution complexities, and even when multiple organizations are involved (as in outsourcing)"
Scaled Agile	ScaledFW	"Disciplined Agile Delivery (DAD) is a people-first, learning-oriented hybrid agile approach to IT solution delivery. It has a risk-value delivery lifecycle, is goal-driven, is enterprise aware, and is scalable." "A common question that we get is what is the difference between primary and secondary roles? Primary roles will occur on all DAD projects regardless of scale. Secondary roles, however, typically occur only at scale and sometimes only for a temporary period of time." "Another common question that we do get is why are there so many roles? For example, Scrum has three roles – ScrumMaster, Product Owner, and Team Member – so why does DAD need ten? The primary issue is one of scope. Scrum mostly focuses on leadership and change management aspects during Construction and therefore has roles which reflect this. DAD on the other hand explicitly focuses on the entire delivery lifecycle and all aspects of solution delivery, including the technical aspects that Scrum leaves out. So, with a larger scope comes more roles. "

		"In one sense methods such as Scrum, Extreme Programming (XP), Kanban, and Agile Modelling (AM) provide the process bricks and DAD the mortar to fit the bricks together effectively." "DAD, because it's not prescriptive and strives to reflect reality as best it can, actually supports several versions of a delivery lifecycle. Six versions of the lifecycle are supported: The Agile lifecycle. This project lifecycle is based on Scrum but extended so as to provide a streamlined strategy from beginning to end. It is depicted in Figure 5 and described in the article DAD Lifecycle – Agile (Scrum Based). The Lean lifecycle. This project lifecycle is based on Kanban. It is depicted in Figure 6 and described in the article DAD Lifecycle – Lean. The Continuous Delivery: Agile lifecycle. This modern agile, stable-team lifecycle is based on Scrum. It is depicted in Figure 7 and described in the article DAD Lifecycle Continuous Delivery: Agile. The Continuous Delivery: Lean lifecycle. This modern agile, stable-team lifecycle is based on Kanban. It is depicted in Figure 8 and described in the article DAD Lifecycle Continuous Delivery: Lean. The Exploratory/Lean Startup lifecycle. This lifecycle is based on Lean Startup strategies. It is depicted in Figure 9 and described in the article DAD Lifecycle Exploratory (Lean Startup). Program lifecycle. This lifecycle is for a team of teams. It is depicted in Figure 10 and described in the article DAD Lifecycle – Program (Team of Teams)."
	ScaledFactors	"One of the great advantages of agile and lean software development is the wealth of practices, techniques, and strategies available to you."

	ScaledPros&Cons	"DAD's goal-driven approach enables DAD to avoid being prescriptive and thereby be more flexible and easier to scale than other agile methods." "There are several fundamental advantages to taking a goal driven approach to agile solution delivery, in that it: • Supports process tailoring by making process decisions explicit. • Enables effective scaling by guiding you through tailoring your strategy to reflect the realities of the scaling factors which you face. • Makes your process options very clear and thereby makes it easier to identify the appropriate strategy for the situation you find yourself in. • Takes the guesswork out of extending agile methods and thereby enables you to focus on your actual job which is to provide value to your stakeholders. • Makes it clear what risks you're taking on and thus enables you to increase the likelihood of success. • Hints at an agile maturity model (this may not be a benefit)." "Enterprise awareness is important for several reasons: You can reduce overall delivery time and cost by leveraging existing assets. In other words, DAD teams can spent less time reinventing the wheel and more time producing real value for their stakeholders. By working closely with enterprise professionals DAD teams can get going in the right direction easily. It increases the chance that your delivery team will help to optimize the organizational whole, and not just the "solution part" that it is tasked to work on. As the lean software development movement aptly shows this increases team effectiveness by reducing time to market."
	ScaledExamples	

Source #22		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	
	OriginFW	"Scrum is the most common agile framework. In fact, it is referred to so often in agile literature that many people use the terms scrum and agile interchangeably."
	ScaleNeed	"Though everyone is going agile these days, a common complaint is that agile development doesn't scale well."
Scaled Agile	ScaledFW	"The Scaled Agile Framework, developed by methodologist Dean Leffingwell, uses a combination of existing lean and agile principles and combines them into a methodology for large-scale projects." "At the Program level, SAFe extends scrum by using the same ideas but one level up. The Program level works on a Release Train, which is composed of five sprint cycles." "SAFe also provides processes one level higher, at the Portfolio level, using lean principles, such as optimizing value streams to help executives and leaders identify and prioritize epics, and features that can be broken down at the Program level and scheduled on Release Trains." "Disciplined Agile Delivery, developed by Scott Ambler and Mark Lines, is similar to SAFe in that it recommends using existing lean and agile techniques. DAD, however, aims to address areas that aren't thoroughly covered in smaller-scale agile frameworks. This framework recommends three phases: Inception, Construction, and Transition."

		"DAD also provides flexibility in suggesting different process guidelines for four categories of lifecycles: agile/basic, lean/advanced, continuous delivery, and exploratory."
	ScaledFactors	
	ScaledPros&Cons	"For this reason, DAD's strengths are in providing more guidance in the areas of architecture and design (inception) and DevOps (transition)." "SAFe has been criticized for being too prescriptive, not allowing teams as much flexibility in process decisions." "Having too much process definition implies a lack of flexibility in approach, which goes against one of the primary agile characteristics of adaptability." "SAFe provides the structure that may make for a smoother transition to an agile framework. SAFe also follows agile practices of inspecting and adapting, even providing an innovation sprint and stressing autonomy and decision-making for the knowledge workers." "With four lifecycle models, DAD provides more flexibility in project guidance and recommendations for best processes within each type of project." "Though this flexibility may be appreciated by those with a good understanding of agile, it might not provide enough guidance for those who are transitioning from traditional models."
	ScaledExamples	

Source #23		
Topic	**Theme**	**Quotes**

Non-Scaled Agile	AgileOriginal	
	OriginFW	
	ScaleNeed	
Scaled Agile	ScaledFW	"SAFe was developed in 2011 to help software development teams bring better products to market faster."
	ScaledFactors	"The biggest benefit of adopting SAFe is the opportunity to tap into a relatively lightweight framework that creates efficiency in software development while maintaining the centralized decision-making necessary at the enterprise level." "…because SAFe was designed to maintain a big picture view of software development, it can easily handle a coordinated strategy for large-scale and complex projects with teams that number into the hundreds." "Another notable benefit of SAFe is its ability to help teams maintain alignment with business goals." "…where methodologies like scrum give more freedom to developers to identify and solve problems that arise due to different sprint cadences, dependencies, etc., SAFe calls for administrative roles to oversee multiple projects and coordinate those releases and dependencies. In taking this freedom away from developers and harkening back to a waterfall environment, SAFe can often slow down processes and limit flexibility compared to an agile environment." "SAFe removes front-line players, including developers, testers and product owners, from the decision-making process and even from one another" "…the fact that SAFe emphasizes the big picture can often lead to longer planning cycles and more fixed roles within development cycles."
	ScaledPros&Cons	

	ScaledExamples	

Source #24		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	
	OriginFW	"In 2016, a leading security products company adopted Scrum to support teams working in complex product development, in order to make the organization more adaptive and able to react faster to change."
	ScaleNeed	"While individually the Scrum Teams were doing really well, meeting their goals and improving their predictability, they faced alignment issues with some non-agile teams, composed of developer, QA, and DevOps roles, which slowed down overall progress." "While the teams were working on one product and had one Product Owner, they suffered from being organized as component teams and dealing with cross-team dependencies."
Scaled Agile	ScaledFW	"SAFe was not an option for them because they believed it was too prescriptive and had too many roles. They didn't have all of the roles and all of their teams worked on the same code base." "Since many of the teams had already implemented Scrum, Venkatesh recommended Nexus as a solution…"
	ScaledFactors	
	ScaledPros&Cons	"By using the Nexus framework, the Scrum Teams have been able to successfully deliver incremental releases, frequently, with minimized dependencies and better alignment across teams." "As a result of using Nexus, the organization has been more adaptive and faster to market than ever before."

	ScaledExamples	"The security products company continues to use Nexus to quicken software delivery while increasing the value they deliver to customers and the organization, helping them remain competitive."

Source #25		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	
	OriginFW	"Prior to implementing Scrum, the company utilized a Waterfall Approach, which did not include iterative or integrated development. This led to inaccurate timeline prediction and fixed-scope releases…" "Net Health first adopted Scrum in 2014 to aid in the completion of two large software development projects and remove the burden of an exhaustive initial requirements phase, thanks to the iterative nature of Scrum." "Net Health's Scrum adoption was initially slow because teams and leaders had to be educated about the value proposition of making the switch, then trained in the new process."
	ScaleNeed	"Once the initial adoption challenges were met, Net Health established five Scrum Teams, over time, all working on large scale projects. The teams were working in silos that didn't efficiently communicate or regularly integrate code, despite the fact that they were all working from the same code base. Each release required massive, complicated merges and significant retesting efforts." "They also needed a way to scale the meaningful conversations they

		were having on individual Scrum Teams for individual products to the whole platform"
Scaled Agile	ScaledFW	"To execute this, and to resolve the issues they were facing with each Scrum Team operating in a vacuum, they selected the Nexus framework, an Agile framework built on Scrum. The Nexus framework drives to the heart of scaling by minimizing cross-team dependencies and integration issues."
	ScaledFactors	
	ScaledPros&Cons	"When Net Health moved to Nexus, they had separate teams of business analysts, with no single empowered Product Owner of their software platform for the Nexus, causing directional confusion." "The Nexus reformed its Scrum Teams through a self-selection team formation exercise and are now delivering integrated increments to their platform." "Over the last two years, Net Health has seen additional teams coming together, more often, with an improved cadence of achieving "Done," and releasing at least one integrated increment at the end of each Sprint." "Thanks to the Nexus framework's focus on integration, code merges are frequent, low-risk, and straightforward." "Teams now regularly evaluate (inspect) each process as they go, identifying those that are higher priority, and speed along only the necessary communications between all stakeholders during the execution of plans. This saves Net Health precious time to make quicker, high value releases and allows the company to respond faster (adapt) to the market."
	ScaledExamples	"With Nexus, Net Health's Scrum Teams are addressing issues head-on through an agile approach, with inspection and adaptation."

Source #26		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	
	OriginFW	"The Cathay Pacific IT team is divided into IT Solutions and IT Infrastructure & Operations. They were primarily utilizing a waterfall approach with some elements of Scrum."
	ScaleNeed	"At that time, they were experiencing a number of challenges with their Software development projects, including long lead times for delivery which had the potential to threaten deadlines. They were only able to release integrated increments once every three months, which was not frequent enough to reach their desired targets." "…there were difficulties in working with some external vendors, meaning a lot of re-work was required on specific projects."
Scaled Agile	ScaledFW	"Nexus is lightweight and simple compared to other scaled Scrum frameworks in the market."
	ScaledFactors	
	ScaledPros&Cons	"With the increase in Product Backlog transparency, it made it easier for the Product Owner to order Product Backlog Items for IBE and other new projects" "The Development Teams became fully aware of the dependencies and began to work more closely with other teams." "Within two months of implementing Nexus for the IBE project, the Scrum Teams increased the frequency of when they were able to release integrated increments by 200%." "Within the Nexus framework, a Nexus Integration Team (NIT) is accountable for ensuring integration across teams during the Sprint, which drove focus toward integration issues and defects to be identified and fixed earlier within Sprint."

		"Another positive development from Nexus was the increased insight of the Product Owner. At first, the Product Owner did not have adequate visibility, and therefore lacked understanding of the delivery progress and what was being developed." "The Product Owner is now able to express what they are accomplishing and delivering in terms of value."
	ScaledExamples	"In February 2017, Cathay Pacific adopted the Nexus framework…"

Source #27		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	
	OriginFW	
	ScaleNeed	
Scaled Agile	ScaledFW	"Nexus and LeSS are the only patterns that are based on real Scrum with 1 Product Owner, 1 Product Backlog for each real customer-facing product." "LeSS's optimizing goal is adaptiveness (agility, cheap & easy change)" "The optimizing goal for Nexus is to deliver better value faster, as opposed to just deliver faster." "Nexus extends the benefits of Scrum to larger and more complex Products (requiring 3-9 teams)." "Nexus is not intended to directly address the broader organizational agility problem that LeSS seeks to solve." "Nexus with Scrum Studio and LeSS have a lot in common. They both deprecate non-Scrum roles. The both have 1 Product Owner and 1 Product Backlog per Product. They are both focused on Agile Product Management. They are both based on real Scrum."

		"LeSS has more upfront asks for the organization, but in essence, it's possible in a huge context to not necessarily start with perfect cross-component teams." "LeSS has an ask to eliminate projects, but in essence, LeSS doesn't expect this to change immediately…" "LeSS is leaning toward the deprecation of single team PBR in favour of multi-team PBR. LeSS has more meat on the bone for what to do with roughly 8 teams or more per product, product definition, coordination, and the overall retrospective including (optional) management" "LeSS requires a journey away from projects." "LeSS is stronger on volunteering, no one gets volunteered." "The LeSS community is very active but less so for ordinary practitioners; it's more aimed at advanced practitioners." "One thing is for sure, if I can't use LeSS for Deep & Narrow, I will at least use Nexus & Scrum Studio" "Of all non-LeSS approaches, Nexus makes most sense and stays best true to Scrum."
	ScaledFactors	
	ScaledPros&Cons	"With adaptiveness, the delivery of better value and the shorter end to end customer cycle times come for free. With adaptiveness, we can "turn on a dime for a dime"." "Nexus has a quicker startup as it can deal with component teams, and evolve towards feature teams, coached by the misnamed Nexus Integration (should be coaching) Team." "I cringe that Nexus still supports projects in the long term, artificial constructs to fund work and create unstable short-lived teams…" "Both options provide a much better alternative to Scrum of Scrums." "I heard a programme director last year who apparently wanted LeSS, that he "didn't care about quality right now", the nub of basic Scrum.

		This is the stuff John Seddon worries about, failure demand from short-term thinking." "And most of all, LeSS has the edge on multi-learning." "Project portfolio management is not needed because projects don't exist in LeSS. Product portfolio might be needed but not if you expand the definition of the product. So portfolio management is not actually a gap in LeSS."
	ScaledExamples	

Source #28		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	
	OriginFW	
	ScaleNeed	
Scaled Agile	ScaledFW	"LeSS: LeSS is Scrum applied to many teams working together on one product. The LeSS Rules are the definition of the LeSS Framework. They are considered a must. Essential SAFe:Essential SAFe is a subset that describes the minimal elements necessary to be successful. If you incorporate these ten essential elements for each release train in your portfolio, you're well on your way to realizing the full benefits of SAFe. Nexus: Nexus is a framework consisting of roles, events, artifacts, and techniques that bind and weave together the work of approximately three to nine Scrum Teams working on a single Product Backlog to build an Integrated Increment that meets a goal." "LeSS

		USE WHEN: There is strong C-level support, you can change org structure, people hate prescription, there is high Agile Maturity in Management and teams START WITH: LeSS Rules, pull in practices from the LeSS Works Framework AND: Provide job safety to Project / Program Managers & consider Daily Scrum for 'shu' teams struggling to self-organize to resolve integration issues." "Essential SAFe USE WHEN: People hate ambiguity and want maximum detail up-front. START WITH: Essential SAFe pull in practices from SAFe Big Picture gradually. AND: Consider adding ART DOD, ART Sprint Planning with team reps, ART Daily Scrum with team reps, ART Backlog Refinement with Team Reps, ART Retro with team reps. Manage mis-alignment between Product Manager & Team level Product Owners. Manage mis-use of I/P Iteration as an enabler for big-bang hardening / stabilization phase." "Nexus USE WHEN: Teams have varying levels of Agile Maturity and are OK with some prescription. START WITH: Nexus Guide, pull in practices from case studies and resources AND: Complement the intentional minimal guidance with practices for your context. Encourage self-study, free practice assessments and online certification to validate shared understanding of framework."
	ScaledFactors	"I believe there should be 5 key desired outcomes for Agile Scaling... VALUE–Scale delivery of valuable features and solutions that customers use QUALITY–Scale quality of integrated features and solutions

		TIME TO MARKET–Deliver valuable features and solutions frequently & regularly WASTE–De-scale the waste of time, effort, money RISK–De-scale the exposure to uncertain, undesirable outcomes" "Based on my experience, most companies face these top 10 impediments to Scaling Business Agility... ACTIVITY BASED SCALING- Focus on scaling # of people, teams, rituals & not on scaling business value PROJECT THINKING–Wasting company time and money on project accounting, context switching SILO'D TEAMS–Teams based on components, functional areas increasing hand-offs, dependencies TEAM PERSISTENCE–Team composition changing frequently, impeding self-organization and learning REACTIVE INTERACTIONS–Reactive, crisis driven interactions between inter-dependent teams LOCAL OPTIMIZATION–Valuing good of the silo over good of the organization ENVIRONMENTS–Lack of shared prod-like environments & data impede frequent integration TOOLS–Ineffective use of tools inhibits interactions, self-organization and learning POLICIES–Org Structure, Hiring, Performance management, Office space, Outsourcing CULTURE–Command and control, waterfall culture inhibits empirical management of value" "Remember that merely choosing an Agile Scaling framework will not magically remove all these impediments. In fact, most frameworks may not even prescribe how to address some of these impediments. They may only provide guiding principles, values and some descriptive suggestions that may help you figure out how to

		make changes to the surrounding eco-system in a way that enables the scaling framework to be effective." "I believe that five key elements of Scaling Frameworks influence the ease of adoption... SIMPLICITY & CLARITY–Simple, clearly expressed framework enables shared understanding BUILDING BLOCKS–Using Scrum as a building block enables Scaled Scrum NEED FOR ORG CHANGES–Minimal addition of new roles, up-front org changes eases adoption EASE OF LEARNING–Free, online resources enable scaled, self-paced learning CERTIFICATION–Certification helps team members validate their understanding" "Now this may come as a surprise to you, but I happen to have some very strong beliefs about what is needed for effective scaling. There are only 15 of them ;), so let me share them with you... A simple, clear framework - Enables shared understanding across teams. Builds on Scrum - Minimize need for teams to un-learn/re-learn. Minimal addition of new roles - Minimizes complexity of explaining new roles, new training, confusion. Customer-Centric, Product Teams - Aligns teams to a shared, elevating true north. Single Product Owner - Avoids disconnects between PO-teams & PO-proxies. Single shared Backlog for all Teams - Creates a single, unifying source of shared work. Shared Definition of Done - Creates single, shared, transparent standard of quality.

		Shared Backlog Refinement with Team Reps - Identifies dependencies, aligns teams, reduces risk. Shared Sprint Planning - Identifies dependencies, creates alignment & plan. Shared Daily Scrum - Enables daily course corrections, daily integration. Single PSI each Sprint - Minimizes risk, reduces cycle time, increases value. Shared Sprint Review - Enables stakeholder feedback & course correction. Shared Retrospective - Enables shared learning, course correction, reduces risk. Ease of learning via free online resources - Enables self-paced learning, shared understanding. Certification detached from training- Enables validation of learning, shared understanding.”
	ScaledPros&Cons	
	ScaledExamples	

Source: #29		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	
	OriginFW	“The survey once again highlighted Scrum as the most popular Agile approach with 66% identifying it as the methodology they follow most closely, with an additional 15% who follow derivations of Scrum (ScrumBan 9% and Scrum/XP 6%).” “Most notably, we have seen the use of Scrum increase from 40% of respondents in the first survey to 66% in the most recent. Other

		approaches such as extreme programming (XP) dropped from use by almost a quarter of respondents to less than 1% today."
	ScaleNeed	"The future requires agile at scale, being able to achieve DevOps, being able to achieve flow, have everybody operating with an agile mindset, having the leaders understand that this is a new way of working — they need to get it."
Scaled agile	ScaledFW	"While a wide range of scaling frameworks are in use, the Scaled Agile Framework (SAFe®) continues to be the most popular with 37% of respondents identifying it as the framework they most closely follow. SAFe® significantly outdistances the next nearest scaling method, Scrum@Scale/Scrum of Scrums (9%)." "Initially, scaling agile was addressed through a "Scrum of Scrums" approach. Over the past five surveys, we have seen the use of SAFe® grow significantly to become the dominant approach, in use by more than a third of respondents"
	ScaledFactors	
	ScaledPros&Cons	
	ScaledExamples	

Source: #30		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	"To increase the speed and flexibility of the services offered to customers, many organizations have embraced agile methods. This started with small projects of interactively and iteratively building and testing software."

		"Small teams of employees work flexibly and respond rapidly to emergent issues and properties."
	OriginFW	"Scrum is the most popular method [49], [50] and has become the de-facto standard for managing knowledge work, especially software development. It is designed for small teams (from three to nine members) who break up their work into so-called "sprints": timeboxed iterations typically lasting two weeks""
	ScaleNeed	"The steady flow of agile-driven endeavours inspired various practitioners to experiment and innovate with the agile methodology, leading to the new challenge of large-scale agile applications."
Scaled agile	ScaledFW	"Three frequently applied ASFs are the SAFe, LeSS, and the Spotify model." "SAFe seems to be broadly applicable, also for organizations with a high degree of complexity. In contrast, the LeSS and Spotify models seem best for organizations with one or more focused service lines. The Spotify model provides very few implementation guidelines."
	ScaledFactors	"It involves critical managerial challenges and consequences for the entire organization [52]. Because of the diversity of organizations, there is no standard roadmap on how to master agile transformations. These authors categorize the challenges for large-scale agile transformations. Given the focus of our study (the governance of the combination of EA and Agile), we address method-, organization-, and culture-related challenges, such as poor customization of agile methods, inappropriate organizational structures, and incompatible social structures." "The challenge of largescale agile transformations is balancing emergent and intentional architecture and communicating and coordinating development activities across teams [15]."
	ScaledPros&Cons	"The benefits of SAFe include improved collaboration and dependency management between agile teams and increased transparency in the organization. Its main challenges are team

		formation, change resistance, organizational politics, and establishing an agile mindset.”
	ScaledExamples	“This resulted in three case companies, two of which are active in financial services and one in telecommunications.”

Source: #31		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	“Initially, they were designed for small, co-located, and self organizing teams that develop software in close collaboration with business customers using short iterations [2]. Hence, agile methods have been primarily applied to projects within the so-called ‘agile sweet spot’, i.e., small and collocated teams of less than 50 persons with easy access to the user and business experts and that develop non-life-critical software [3].”
	OriginFW	“Within few years, various agile methods appeared on the landscape, such as Extreme Programming and Scrum, to name a few [1].”
	ScaleNeed	“The success of agile methods in small, co-located projects has inspired companies to apply them in larger projects.” “Given the successful adoption of agile methods in small organizations and projects, also many large software organizations have begun to adopt these methods” “The successful adoption of agile methods in small teams ignited a new passion among firms to start using agile methods in large projects, even beyond software development, across the enterprise [11]. This phenomenon is often referred as ‘large-scale agile development’ [12].”
Scaled agile	ScaledFW	“Agile scaling frameworks, such as Large Scale Scrum and Scaled Agile Framework, have been invented by practitioners to scale agile to large projects and organizations.”

		"To resolve issues associated with the adoption of agile methods in large-scale organizations and projects, several agile scaling frameworks, such as Large Scale Scrum (LeSS)1 and Scaled Agile Framework (SAFe)2, have been created both by some custodians of existing agile methods and by others who have worked with companies to scale agile methods to their settings [4,6,7]." "For instance, Alqudah and Razali [10] juxtapose DAD, LeSS, Nexus, RAGE, SAFe, and Spotify based on, e.g., team size, available training and certificates, and the underlying agile methods and practices. Diebold et al. [18] provide a map visualizing underlying agile practices of different frameworks, such as DAD, LeSS, and Nexus, to support organizations in the selection of appropriate frameworks. Based on 13 agile transformation cases, Conboy and Carroll [16] provide nine challenges and a set of recommendations associated with agile scaling frameworks, such as LeSS, Nexus, S@S, and Spotify." "According to our survey data, Crystal is the first created agile scaling framework which development started in 1997. Nexus, eScrum, and S@S were also relatively early designed compared to most other agile scaling frameworks." "Most frameworks have multiple versions, whereas four frameworks have only one version, namely Nexus, LeSS, Spotify, and XSCALE. Gill was initially created as the ASSF framework (2005-2008), which then evolved into Gill in 2012. The methodologists of Mega indicated that Mega 1.0 was a derivative of SoS. They also stated that Mega 2.0 was influenced by Spotify including the idea to extend the adoption of agile practices to other parts of the organization. The methodologists of Spotify found inspiration from Craig Larman's and Bass Vodde's two books (cf. [22,23]), that later became LeSS. Spotify was also influenced by the Program Increment Planning events of SAFe (cf. [24])."
	ScaledFactors	"The most mentioned challenges were using frameworks as cooking recipes instead of focusing on changing people's culture and mindset."

		"However, the adoption of agile methods outside the agile sweet spot poses significant challenges to organizations, such as coordination challenges in multi-team environments [5]." "First, the scaling of agile methods entails additional scaling and complexity factors that summon 'bitter spot' conditions for agile methods, such as a large number of teams, geographical distribution, entrenched culture, or formal governance structures [14]. Second, present agile methods do not provide sufficient guidance on dealing with these scaling and complexity factors [15]." "The most commonly stated reasons were: improving the agility/adaptability of the organization, improving the collaboration of agile teams working on same product, improving the coordination of agile teams working, and improving the synchronization of agile teams working on same product." "These benefits were grouped into two categories, namely: business/ product and organization/culture. The most commonly mentioned benefits were: enabling frequent product deliveries, enhancing employee satisfaction/motivation/engagement, improving software quality, providing customer/business value, improving the collaboration of agile teams working on same product, improving the coordination of agile teams working on same product, improving the synchronization of agile teams working on same product."
	ScaledPros&Cons	* Table 3 in source shows benefits per scaled agile framework. * Table 4 in source shows challenges per scaled agile framework.
	ScaledExamples	

Source: #32		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	

	OriginFW	
	ScaleNeed	
Scaled agile	ScaledFW	"In the Scaled Agile Framework, SAFe (Leffingwell 2018), teams are required to work according to a common high level rhythm, called the Program Increment, which typically lasts about 12 weeks. The teams can work according to either ScrumXP, an amalgamation of Scrum (Schwaber and Sutherland 2017) and XP (Beck 2000), or team-level Kanban. In the LeSS framework (Larman and Vodde 2016), all teams follow the Scrum process with a synchronized rhythm. In the discussion, we contrast these approaches with our findings in more detail." "SAFe represents the most traditional model, with the highest level of standardization and management responsibility and subsequently the lowest degree of team autonomy." "The LeSS model has a different take on management, stressing the need for managers to facilitate the development teams, and taking responsibility for systemic improvement of the development organization, but letting the teams and product owners take care of all product decisions, as well as process improvements (Larman and Vodde 2016)." "The Spotify model, also based on Scrum, has a high level of team autonomy, as the teams are involved in or are able to influence all kinds of decisions, from executing the task to setting the overall direction."
	ScaledFactors	"Further, when implementing a shared decentralized decision-making process in large projects with many teams, alignment problems must be tackled. Previous studies have addressed the challenges associated with the lack of alignment among teams (Bick et al. 2017). If each team works independently, team development processes and technical solutions will ultimately differ and may be highly disconnected from one another. Further, the absence of knowledge sharing between teams can lead to duplicated work, misunderstandings, and integration problems. However, when practices in teams become dependent upon

		other teams' practices, teams can hardly be described as autonomous (Rolland et al. 2016)." "In a literature review of the challenges and success factors of large-scale agile transformations (Dikert et al. 2016), team autonomy was mentioned as both a challenge and a success factor. On one hand, grassroots-level empowerment was highly important for transformation success. On the other hand, team autonomy was challenging to arrange in large-scale settings. In particular, challenges arose if the organization had developed their software development model according to agile principles, and thus allowed teams to operate independently while dependencies existed between the teams, requiring constant collaboration. Further, it was challenging to balance the team goals and the broader goals of the organization. A team could easily place more emphasis on their own goals over the goals of the whole organization. Thus, being part of a large-scale organization might limit the amount of authority that can be given to each team."
	ScaledPros&Cons	
	ScaledExamples	"In this paper, we report our findings from a multiple-case study in two large-scale software development organizations in the telecom industry." "Perhaps most saliently, the approach used by Spotify (Kniberg 2014a, b; Mankins and Garton 2017) is built around the idea of autonomous agile teams aligned through common goals and a number of bottom-up coordination mechanisms, including communities of practice called guilds (Smite et al. 2020, 2019). The agile coaches at Spotify attempt to balance autonomy alignment across teams (Bäcklander 2019)." "To understand how authority is allocated between the teams and the rest of the organization, we designed a multiple case study of two large-scale agile software development projects at Ericsson, a global leader in telecommunications systems." "Starting in the spring of 2009, the organization transitioned to a "lean and agile" way of working, in their own words and taking inspiration from the Large-Scale Scrum (LeSS) scaling framework (Larman and

		Vodde 2016). The main reason for the transformation was the need to shorten the development cycle and build capacity for more rapid reactions to market and customer needs. The development process was Scrum-based, with an iteration length of two weeks."

Source: #33		
Topic	**Theme**	**Quotes**
Non-Scaled Agile	AgileOriginal	
	OriginFW	"FOR the last two decades, agile methods such as Scrum[1], [2], eXtreme Programming (XP) [3], DevOps [4], and low [5] have proven overwhelmingly popular and, for the most part, highly effective amongst software development teams."
	ScaleNeed	"Reasons underpinning the emergence of these large-scale1 methods include a need for alignment and cohesion across many teams, deep interdependencies between software development and other organisational functions such as human resources, legal, and finance, as well as the global trend toward large distributed teams and product delivery at scale [6]."
Scaled agile	ScaledFW	"Over the last number of years, quite a few large-scale agile methods have been proposed, e.g. the Scaled Agile Framework (SAFe) [12], and Large-Scale Scrum (LeSS) [13]." "The number of these types of studies did not diminish despite the emergent and prevalence of large-scale frameworks such as SAFe." "For example, SAFe explicitly mentions four core values (alignment, built-in quality, transparency and program execution), Lean-Agile mindset and ten principles. LeSS has ten principles. DAD has seven principles, while Scrum-at-Scale defines five core values (openness, courage, focus, respect, and commitment) and Spotify model has ten. In this study, we grouped the terms value, mindset, and principle together under principle, as they inspire and inform the practices, tools

		and metrics of a framework. From this point on only the term principle is used."
	ScaledFactors	"These scaled-up methods have struggled to deal with the exponentially greater complexities and interdependencies of largescale, organisation-wide development. The difficulties usually centre around the complexities and ambiguities, arising from (i) a large number of teams, roles, and personalities; (ii) an often unknown composition of participant teams and projects at the outset; (iii) abstract, knowledge-intensive, and often ill-structured work processes; (iv) diverse and often competing agendas between teams that sometimes contradict the organisation itself; (v) abstract, complex, and often unknown final outputs and goals [9], [10], [16]." "Literature shows that large-scale development involves challenges related to inter-team coordination, large project organisation, release planning and architecture, customer collaboration (including contracts), and knowledge sharing and improvement [55]."
	ScaledPros&Cons	"For improving inter-team coordination, the Scrum-of-Scrums (SoS) meeting is the most common connecting practice reported in 17 primary studies of 15 case organisations), in which Scrum masters from each team meet to coordinate the delivery of software and solve inter-dependencies." "SoS meetings may not be an effective communication channel to discuss and address impediments (PS36). One reason is due to the multiple agile layers (C-IC-2). As the number of layers grows larger, meetings take longer to cover all topics and teams may not receive any feedback or solutions to their problems (PS49, PS96)." "This is also the case for SAFe. Spending too much time in joint meetings, e.g. Product Increment (PI) Planning meetings, was considered waste." * Table 8 in the source gives challenges based on specific scaled agile frameworks

		"SAFe was designed originally for scaling agile practices in large organisations. Compared to other large-scale methods, it has more roles, events, artefacts, and practices. However, focusing excessively on adopting and tailoring these elements could distract an organisation from achieving its business goals (PS104, PS105, PS150)." * Table 9 in the source gives pros of the scaled agile frameworks
	ScaledExamples	