Lappeenranta University of Technology

School of Engineering Science

Degree Program in Computer Science

Niklas Nygren

FACTORS AFFECTING AGILE TEAMS' PERFORMANCE

Examiners:	Professor Jari Porras
Supervisors:	Professor Jari Porras
	Sanna Repo

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
School of Engineering Science
Tietotekniikan koulutusohjelma

Niklas Nygren

Tiimien tehokkuuteen vaikuttavat tekijät ketterässä kehityksessä

Diplomityö

2018

65 sivua, 11 kuvaa, 6 taulukkoa

Työn tarkastaja: Professori Jari Porras

Hakusanat: ketterä kehitys, SAFe, tiimien tehokkuus

Tässä diplomityössä suoritettiin tutkimus seikoista, jotka vaikuttavat Agile-tiimien tehokkuuteen SAFe-viitekehystä käyttävässä organisaatiossa. Tutkimus tehtiin yhdellä Accenturen asiakkuudella ja se on rajoitettu tiimeihin, joiden kaikki henkilöt työskentelevät Accenturella. Tutkimusaineisto kerättiin tiimien kevääseen 2018 mennessä keräämistä Sprint-materiaaleista. Kerätystä aineistosta poimittiin asiat, joilla oli vaikutusta tiimien tehokkuuteen ja ne jaettiin kirjallisuuden pohjalta valittuihin kategorioihin. Aineistosta löydettiin 366 tiimien tehokkuuteen vaikuttavaa tekijää, joista 180 olivat positiivisia ja 186 negatiivisia. Suurin positiivinen vaikutus tiimien tehokkuuteen oli hyvä tiimin sisäinen koordinaatio, tavoitteellisuus ja jatkuva kehittyminen. Tehokkuutta laski eniten tiimin ulkopuolelta tulleet häiriötekijät: riippuvuudet ja tekniset ongelmat. Kehittämällä tiimin koordinaatiota ja suunnittelua sekä minimoimalla ulkoisten seikkojen negatiivista vaikutusta tiimien tehokkuus kasvaa.

ABSTRACT

Lappeenranta University of Technology

School of Engineering Science

Degree Program in Computer Science

Niklas Nygren

Factors affecting Agile teams' performance

Master's Thesis

2018

65 pages, 11 figures, 6 tables

Examiner: Professor Jari Porras

Keywords: agile development, SAFe, team performance

This thesis conducts a research into factors affecting Agile teams' performance in an organization using Scaled Agile Framework (SAFe). Research was conducted at one Accenture client and is delimited to only Agile teams which have all members employed by Accenture. Research data was gathered from Sprint retrospective material created by the Agile teams up to spring of 2018. From the data, factors that had impact on team performance were identified and assigned to categories based on the literature composed. 366 items affecting team performance were found from the data of which 180 had a positive impact and 186 had negative impact. Biggest positive impact to the team came from internal team coordination, goal orientation and continuous learning within the team. Factors having the biggest negative impacts on performance were from external factors; dependencies and technical issues. Improving team coordination and planning and reducing the negative impact of external factors will improve Agile team performance.

ACKNOWLEDGEMENTS

I would like to thank Accenture Finland for allowing me to conduct this research.

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

ART	Agile Release Train
FDD	Feature-Driven Development
LSD	Lean Software Development
PO	Product Owner
RTE	Release Train Engineer
SAFe	Scaled Agile Framework
SLR	Systematic Literature Review
XP	Extreme Programming

1 INTRODUCTION

Software development methods are evolving all the time to meet the demanding needs of companies everywhere. Especially agile and lean agile methods are pushing aside the more traditional methods such as Waterfall. This thesis focuses on a software development method called Scaled Agile Framework (SAFe) with focus on version 4.0. It takes advantage of the Agile methods and applies them to the enterprise scale. SAFe is designed to provide guidance on an enterprise level on how to achieve the benefits of the Lean-Agile development and to deliver value continuously on a predictable schedule. This leads to a competitive edge in the marketplace for the organization. SAFe is divided to four levels: Portfolio, Value Stream, Program and Team level. In this thesis, the focus is on the Team level which consists of Agile teams that together form an Agile Release Train (ART). It is a self-organizing team of Agile teams (about 50-120 people), which work together toward a common mission. The different roles in an ART work seamlessly together to further its objectives. Every role has a part to play in the overall effectiveness of the Agile Release Train. (Scaled Agile Inc, 2016)

1.1 Background

Accenture is a leading technology consulting company operating worldwide in over 120 countries. It has clients in over 40 industries. In Finland Accenture has operated since 1989 and it employs approximately 1400 people. Accenture provides application development services by using different kind of frameworks based on the client's needs.

Using SAFe and other agile development methods has increased in recent years in Finland. According to experiences of SAFe usage in application development there are some common issues within organization which are the obstacles to improving Agile team performance and increasing predictability of value creation for the clients. Objective is that with this research those issues can be identified, and the results can be used in planning on how to make improvements. (Accenture, 2018)

This thesis focuses on the Team level structure of the framework and conducts a research into the different factors that affect team performance. What kind of different factors affect the performance of Agile teams and how they can be mitigated/improved is the goal of this research

1.2 Goals and delimitations

Goal of the research is to find out what are the different factors that affect Agile team performance in an organization using SAFe. In addition, the research goal is to find out how team performance can be improved. The research is limited to the Agile teams operating within the Scaled Agile Framework on one Accenture client and to teams which have all members employed by Accenture. Multivendor teams were not included. This delimitation was chosen because the research would be too large in scale to execute and to guarantee that all included teams have the same development environment. Aim of the research is to find answer to the following question:

- What are the factors affecting the performance of Agile teams in SAFe model delivery?

The execution of the research is explained in the next chapter, which highlights the limitations.

1.3 Research method

The literature section aims to find the different theoretical items that affect agile team performance from within and outside the team. The origins of the SAFe method are explored by going through Agile and Lean methods. Components of teamwork and factors affecting team performance are reviewed to identify the internal factors affecting team performance. Based on the literature assembled categories are chosen where the factors found from the research are assigned to. The research material is gathered by going through data from Sprint Retrospectives recorded by the Agile teams at the end of each sprint. After finding the most common factors that negatively/positively affect the team performance we aim to find solutions which can mitigate the negative causes and increase the number of positive factors. The research will be done for Accenture Finland internal development purposes.

1.4 Structure of the thesis

Section 2 contains a description of the relevant literature concerning the research. Agile and Lean Agile methods are discussed and also relevant parts of the Scaled Agile Framework for the research scope. In addition, factors affecting the team performance are also gathered.

Section 3 presents the execution of the research. It describes how the research is conducted and how the data found from Sprint Retrospectives is categorized into theory-based categories.

Section 4 presents the results of the research and the analysis of said results.

Conclusions are presented in the section 5. Section 6 summarizes the thesis, what was done and what was found.

2 LITERATURE REVIEW

The literature section aims to find broad categories for factors that affect the performance of Agile teams which are then used in the research to assign the found specific items to these categories. The research is concentrated on Agile teams working within the SAFe-model so the literature section presents the origins of said model to the reader by going through Agile methods, Lean Software Development and finally Scaled Agile Framework. Literature about the Scaled Agile Framework is combined only from material provided by Scaled Agile Inc as it is a trademark of said company and that is the model that is being used by the teams in the research scope. Categories for factors affecting team performance from outside the team are based on this part of the literature review. By identifying components of teamwork in the Team Performance-chapter, categories for factors affecting team performance from within the team can be defined.

2.1 Agile methods

Agile was originally taken to use in the manufacturing sector and over time it has evolved into a set of practices and principles that have found success in the IT sector and beyond. It enables and encourages innovation with its iterative and incremental manner that empowers employee through organizational and experiential learning. Many of its methodologies have established themselves in different facets of Agile from engineering, product development, project management and enterprise architectural perspectives which are also being influenced by other development methods (Moran, 2015. Some of the popular Agile methods used in the software industry are Scrum, Extreme Programming (XP) and Feature-Driven Development (FDD). (Markkula et. al, 2011)

As a concept, Agile was brought to the forefront of development processes in the late 1990s as an answer to existing solution development processes that were rigid, hierarchical and plan-driven, such as the Waterfall model. It became more prominent with the rise of new technology and the increasing change in the business environment. Agile was promoted with the notion that uncertainties in projects should be accepted as inevitable and they should be embraced. It advocated for execution and feedback to be balanced with planning and control (Moran, 2015). Generally agile software development can be characterized as incremental, cooperative, straightforward and adaptive, (Choudhary et al, 2016). Agile values were put together in the Agile Manifesto in 2001 by seventeen independent software developers. They express the agile way of preference towards individual interactions and customer collaboration, working solutions over comprehensive documentation and ability to response to change over following a plan. The Agile Manifesto is presented in picture 1 below. (Agile Alliance, 2015).

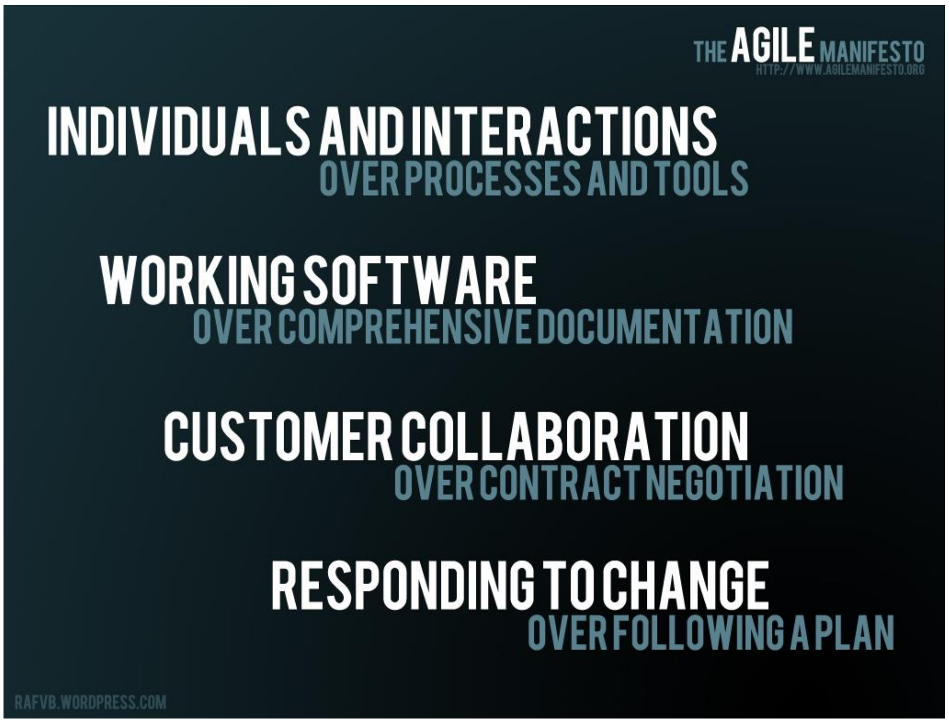

THE AGILE MANIFESTO
HTTP://WWW.AGILEMANIFESTO.ORG
INDIVIDUALS AND INTERACTIONS
OVER PROCESSES AND TOOLS
WORKING SOFTWARE
OVER COMPREHENSIVE DOCUMENTATION
CUSTOMER COLLABORATION
OVER CONTRACT NEGOTIATION
RESPONDING TO CHANGE
OVER FOLLOWING A PLAN
RAFVB.WORDPRESS.COM

Figure 1. The Agile Manifesto (Van Baelen, 2016)

The manifesto puts forward the belief that interactions among the team members and their customers should support the implementation of working solutions in a highly adaptive manner. In addition to the values expressed in the manifesto, equally important to defining Agile are the twelve principles behind it. They are presented in the table below.

1	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2	Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.
3	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4	Business people and developers must work together daily throughout the project.
5	Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
6	The most efficient and effective method of conveying information to and with a development team is face-to-face conversation.
7	Working software is the primary measure of progress.
8	Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9	Continuous attention to technical excellence and good design enhances agility.
10	Simplicity – the art of maximizing the amount of work not done – is essential
11	The best architectures, requirements and designs emerge from self-organizing teams.
12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Table 1. The twelve principles behind the Agile Manifesto. (Agile Alliance, 2001)

Agile cannot be purely defined by its principles, but it can be understood as a structure for solution development that is surrounded by a regular cadence of iterative development cycles and incremental delivery that is driven by the business needs of the customer. The following core elements work best in tandem in promoting the agile way of working.

- *Adaptive*. Per Agile change is inevitable and the possibility of risks must be accepted when reaching for rewards. Adaptive planning and effective feedback loops supported by decentralized and iterative approach are the culmination of Agile development in the constantly changing business environment.
- *Value Driven*. Focus on business needs by measuring working solutions rather than status reports, Agile promotes a more direct assessment of progress. It concentrates on the direct responses and evaluations from the customer. Close integration of stakeholders is required in addition to lean production practices and incremental delivery of working products.
- *Collaborative*. Multi-disciplinary and highly communicative teams are preferred instead of specialists working on specific parts of the solution. The team, by way of communicating with each other gains knowledge regarding the consensus of the solution. Outside stakeholders may be needed to help the team in specific areas.

- *Empowered*. The team works together towards a common goal which requires trust and respect between the team members, supported by an environment of empowerment and self-organization powers the team to better efficiency. Instead of a traditional management role the team is managed by one its members in a servant-leadership role, by helping the other members in every way he/she can.

What can be derived from the core elements is that agile isn't just a set of practices and values but also a cultural stance on the process of solution development. Agile regards uncertainty during the development process as a strength as it allows for new ideas to come forward instead of following a strict plan. Many agile principles are supported by existing management theories. For example, building projects around motivated individuals and promoting sustainable pace finds itself embedded in the agile notion that employee's motivation is increased by working toward outcomes which in turn lead to rewards. (Moran, 2015)

Benefits of Agile

Benefits of agile software development have been well documented in different settings and they can be categorized to strategic and tactical advantages from the business point-of-view in the following manner as presented by Cooke (2012):

Strategic benefits:

- **Constant risk management:** By constantly communicating with the customer during the development process, requirements can be confirmed and adjusted so that technical risks are mitigated as soon as they are discovered.
- **Constant management of budget:** Business value of the deliverables can be reviewed after every iteration which gives the customer the possibility to adjust, postpone or stop development on a certain item if the business benefits do not seem to be on par with the investment.
- **Working software delivered in fast cycles:** Development teams focus on developing fully functional, fully tested and usable software in short iterations, rather than over a longer period.
- **Software aligned with business requirements:** Constant communication between the developers and the customer during the development process allows feedback to be incorporated to the software as it is being developed.

- **Highest priority features are prioritized:** Continuous communication with the customer allows the organizations priorities regarding the features in development to be kept up to date.
- **Adaptiveness to change:** Changes in the organization, industry or technologies can be adjusted and incorporated during the development process.
- **Transparency to the customer:** Regular releases of working software for the customer and the constant communication allows for greater transparency between the developers and customer.
- **Higher quality software:** Continuous testing during development and constant communication with the customer regarding the business value of features being developed leads to high quality deliverables.
- **Greater employee satisfaction:** Work environment that is based on high communication among team members, empowerment of the individual, encouragement of innovation and regular acknowledgement of contributions achieved greater employee satisfaction.
- **Minimized software costs during its lifecycle:** Development costs and support and support and maintenance overheads are provided by incorporating usability, quality and extensibility to products during the development process. Also, responsiveness to change makes it less-risky and more cost-effective to add additional functionalities to the products.

The strategic benefits presented above can also deliver *tactical* benefits such as:

- **More production per resource hour on valuable outputs:** Several of the strategic benefits including *highest priority features are prioritized, constant risk management and higher quality software* can significantly increase the effectiveness of the employees working time. This amounts to more business value for the same level of effort, same timeframe and same budget. Higher productivity also minimizes the need for last second hurry and possible overtime as the standard working hours are used to maximum effect.
- **Technical issues are spotted earlier:** *Working software delivered in fast cycles* and *constant risk management* leads to production of fully functional and fully tested features. The development of these functional capabilities requires several working

system components including established architecture, working database and a functional integration platform. All these can assist in spotting technical risks as early as possible before they become major issues.

- **Minimized amount of rework and "throw-away" work**: *Software aligned with business requirements, highest priority features are prioritized* and *adaptiveness to change* advocate for the staff to focus on producing capabilities that are actually needed by the business and with features that are going to be used by the customer. It creates a stronger working relationship with the customer while also making it less likely that there are some critical features urgently needed to be added to an already delivered system.
- **Reduced need for detailed documentation:** *Working software delivered in fast cycles, software aligned with business features* and *adaptiveness to change* all advocate for the organization to reduce the dependency for detailed documentation of requirements before the development process can begin. By quickly learning to trust the process and the development team to deliver wanted capabilities, the business users will not need to rely on pre-defined documentation as a "safety net" for ensuring their needs will be met. Also, *transparency to customer* makes the development team less dependent on detailed project plans as a way of making sure that work keeps on track to deliver the agreed products.
- **Increased flexibility:** *Working software delivered in fast cycles* and *adaptiveness to change* both account for the fact that the solutions need to be easily extensible to minimize overheads when changes are at some point needed. The flexibility of Agile approaches doesn't end with technical flexibility; management is provided flexibility by the incremental planning process that is included in the *constant management of budget*, by allowing people in charge to regularly review and adjust work so it meets the highest-priority demands of the department.
- **Increased staff autonomy:** *Constant risk management, working software delivered in fast cycles* and *transparency to the customer* all account for the fact that agile work is visible to management without the need for constant supervision. In addition to the benefits from *software aligned with business requirements* means that the work done is continuously reviewed and confirmed by the customer. All these allow for both the staff and the management to better focus on their most important responsibilities, which can lead to more efficient use of limited resources.

- **Higher job satisfaction:** *Working software delivered in fast cycles* gives the staff a greater sense of accomplishment in their work than they could get from producing a lot of documentation. This added to the benefits from the delivery of *higher quality software* results in employees to take greater pride in the work they do. In the big picture, this can lead to *greater employee satisfaction*, with the added benefit of reducing the department's overhead in replacing employees and the preservation of corporate memory.
- **Increased innovation opportunities:** An environment where *working software delivered in fast cycles* along with production of *higher quality software* while also having good *adaptiveness to change* is a good situation for innovation as employees are required to be creative as they work to develop solutions that have adaptive functionalities. Innovative and cutting-edge solutions can arise from such an environment.
- **Less dependency on status reports:** *Working software delivered in fast cycles, constant risk management* and *transparency to the customer* combined reduce the dependency on the staff to produce status reports to higher management. Progress is measured by the production of working solutions and the customer's response to said solutions along with real-time tracking tools.
- **Less need for support and maintenance:** *Software aligned with business requirements, highest priority features are prioritized, adaptiveness to change* and *higher quality software* all amount to minimizing the software costs for the delivered solutions for their entire lifecycle. Before solutions are released into production they are fully tested and confirmed by the business users. Completely error free software is a utopian thought along with that future changes won't be needed, as the business demands continuously evolve. However, the extensive testing and quality control during the development process and highly adaptable solutions can greatly reduce the needed time and cost for maintaining live solutions.

Problems in the adoption of Agile methods

To gain the benefits of the agile the organization must succeed in the adoption of the agile software development methods. The adoption contains many difficulties and a Systematic Literature Review (SLR) conducted by López-Martinez et al. (2016) categorized the found problems in four categories: people, processes, project, and organization aspects of the company. Most problems were found in the people category. Some of the problems in each category are presented next with an overview of the category itself.

People

Change from traditional plan-driven development environment to an agile method requires changing the mindset and thinking of the employees which is a critical factor in how successful the migration is. Most of the problems involve lack of effective communication as more informal communication can lead to improved success in the project. Some of the problems found in the SLR are the following: size of the team, general resistance to change, lack of experience with agile methods and lack of collaboration and communication with the customer.

Processes

Processes are essential to agile methods, such as Scrum. Problems that were identified in the study were agility decree and anti-patterns.

Project

Project category includes customer satisfaction, project cost, duration, size and complexity. Majority of the papers found in the study were focused on organizational aspects but some of the issues found regarding the project were size of the project and the difficulty to scale in larger projects.

Organizational aspects

Organizational culture has great impact on the adoption of agile methods as they must be used within a culture that supports their use. A culture that has a capacity for change, broad support for negotiations and collaboration and continuous exchange of knowledge among its employees is a necessary aspect in a successful adoption of agile. Some of the problems that organizations have in the adoption of agile methods are the following: organizational culture doesn't support agile work, it is lacking the capacity for changing the culture, organizational problems, lack of support from higher management and external pressure to use traditional development practices.

Scrum

Scrum is the most popular agile methodology used in the software industry. It is a framework that is easy and simple to understand while being capable to manage the production of complex products. The framework consists of Scrum team and their activities, artifacts and rules. The team consists of a product owner, scrum master and development team of about 5-9 people. The product development is done in short cycles called sprints which have a set duration, typically from 2 weeks to a month.

Product owner is the person in the Scrum team with the vision about the product. He is responsible for getting the most out of the team by making sure the problem being solved is divided accordingly among the team. Scrum master leads the development team and acts as a servant leader, helping and guiding them while resolving issues and removing impediments. He is also responsible for conducting the daily meeting where the status of the project is gathered from the team members. The development team consists of highly skilled developers who are vital in achieving the goals of the project and making sure there is something to release at the end of each sprint.

Essential for scrum is the product backlog which contains a prioritized list of product features that the product owner has created according to his vision of the product. A sprint starts with sprint planning which includes updating the spring backlog which contains the work to be done during the sprint by the team. Products features are taken from the product backlog into the sprint backlog if the team thinks they can complete it during the sprint. In the sprint backlog the features are broken down into tasks that are then done by the team during the sprint. Daily scrum meetings are organized by the scrum master to gain visibility to the direction and progress of the project. Sprint review is conducted at the end of each sprint where the product is inspected by all stakeholders. They give feedback about the product and update their vision to the team if necessary. Sprint retrospective is held at the end of each sprint after the sprint review and before the next sprint planning. Sprint retrospective is focused on improvements that the team can do to increase their success in the next sprint cycle. The Scrum framework and sprint cycle is presented in the following picture. (Sharma and Hasteer, 2016)

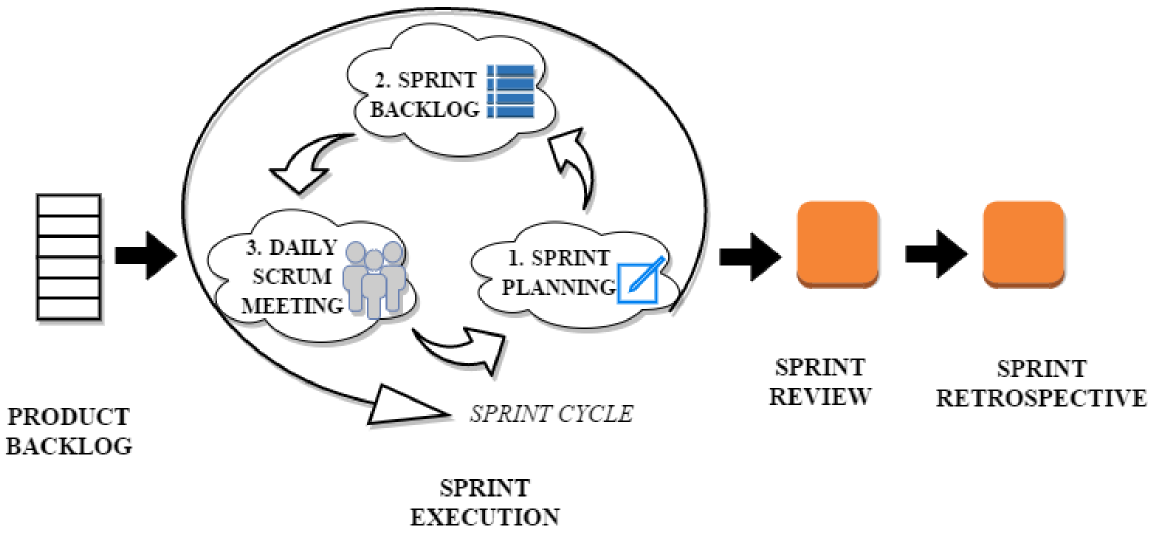


Figure 2. Scrum Framework. (Sharma and Hasteer, 2016)

Extreme Programming (XP)

Extreme Programming is an agile software development methodology that reduces risk, improves adaptiveness to changing requirements while allowing a system to naturally grow in an enjoyable working environment. It is a combination of a set of industry best practices customized to a changing environment where the customer is assigning a moving target.

XP consists of four values and twelve practices that are derived from the values. The values are the following:

- *Communication:* Personal communication is favored over impersonal communication. It is the preferred communication choice in XP, but it does not mean that documentation is ignored or not produced when necessary.
- *Feedback:* Straightforward feedback is more useful than wishful optimism or hopefulness. Working code gives better feedback than documents or models about the state of the project.
- *Simplicity:* Small, simplest parts of the solution that can be done are valued over complex ideas that try to plan for the future. Parts of the system that are known and can be developed are done in the simplest way possible and then modified when new changes are required due to changing requirements.

- *Courage:* Developers can choose how they want to complete their work as the system design is the simplest possible and there are always automated tests that can be used for validation.

The practices that are derived from these values are the following: planning, small releases, metaphor, simple design, testing, refactoring, pair programming, collective code ownership, continuous integration, 40-hour week, on-site customer and coding standard. (Pyritz, 2003)

Feature-Driven Development (FDD)

Feature-Driven Development is an agile methodology process designed to help teams produce tangible results in a predictable schedule. It is based on using small pieces of client-valued functionality, called features. The features are organized into business-related sets. FDD uses a two-week development cycle that produces results. Development with FDD consists of five processes that are presented by Coad et al. (1999).

1. Development of an overall model using starting requirements and features. Focus on shape of the overall system.
2. Building of a detailed and prioritized features list.
3. Planning for each feature.
4. Design for each feature using different components and focusing on sequences.
5. Develop by feature.

FDD has two key roles: chief programmers and class owners and an important structure: feature teams. Chief programmers lead the feature teams in developing a specific feature. They lead by example and help the team in completion of their tasks. Features are assigned to chief programmers who then form a temporary feature team in order to complete that feature. Class owners are responsible for designing and implementing the different classes. Usually one class owner has ownership or one class. Class owners work on more than one feature team at a time. Their feature-team membership may change with each iteration.

Most of the time in FDD is spent in the processes 4 and 5, designing and developing the features. After the planning for features has been completed the process continues iteratively in 2-week cycles on the design and development of the features.

2.2 Lean Software Development

Lean was originally a manufacturing process introduced by Toyota around the halfway point of 1900. It concentrated on minimizing overburden and inconsistency as well as the elimination of waste. In the 1990s it became known as Lean Manufacturing or Lean for short. It follows the five principles of Lean which are: *Value*, *Value Stream*, *Flow*, *Pull* and *Perfection*. *Value* is the basis of any product and understanding what adds value for the customer. *Value Stream* includes everything that is required in bringing the project from start to finish. All steps that do not add value should be removed. Achieving *flow* maximizes speed and minimizes the costs as the development process is in motion all the time. Continuous flow promotes a more efficient process that leads to quicker and higher quality products. *Pull* starts all lean development cycles. It means that nothing will be made unless it has been requested by the customer. This way no unnecessary waste is produced because of products that are not actually requested by the customer. Continuously aiming towards *perfection* is the fifth principle of lean. Lean development process is continuously examined to optimize it even better. (Womack and Jones, 2003)

In the early 2000s Lean was taken to use in other forms of manufacturing and development. The Agile manifesto which relates to software development shows similarities with the principles of Lean. Lean Manufacturing was modified by Mary and Tom Poppendieck to Lean Software Development (LSD) which describes how it can be used in a software development environment. (Jewett, 2008)

Lean model that was modified by Poppendiecks (2003) includes the following principles:

- Eliminate waste
- Amplify learning
- Decide as late as possible
- Deliver as fast as possible
- Empower the team
- Build integrity in
- See the whole

Eliminate waste

Elimination of waste is one of the most essential principles of Lean Software Development. Waste is anything, a task or a process, that does not directly produce value to the end customer. Interpreted to software development, it is slightly different as most forms of waste are intangible. Things that could be categorized as waste include specific processes, code, files, data, features and comments. The most difficult part in eliminating waste is locating waste that can be eliminated.

Waste can be found in seven places according to Mary Poppendieck called “Seven Wastes of Software Development” which are:

- Partially Done Work
- Extra Processes
- Extra Features
- Task Switching
- Waiting
- Motion
- Defects

Flow of value to the end customer is interrupted by each of these wastes. *Partially done work* doesn’t give any value as it hasn’t been completed yet. It could be waiting for things to be added but before that it is not giving any value.

Extra processes that do not any add value should be eliminated as they are considered waste. These can be for example unnecessary paperwork that doesn’t produce any customer value even though it be required by the customer. In cases like these the following rules should be remembered: Keep it short. Keep it high level. Do it off line.

Extra features are parts of software that the end user does not have any use for, thus they are considered waste and should be eliminated. Very often features that are developed are never used or used rarely by the end customer as found by a study by the Standish Group shown below.

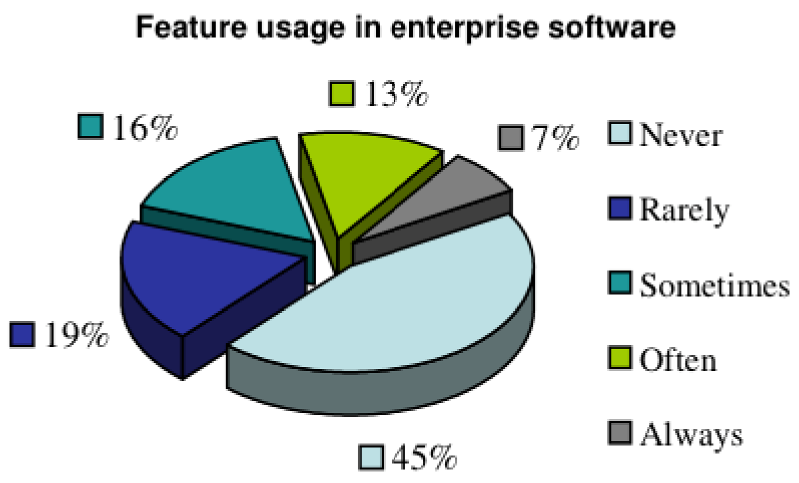


Figure 3. Study by Standish Group showing feature usage in enterprise software. (Johnson, 2002)

According to Mary Poppendieck the unused features should be the first thing to look when locating waste that can be eliminated.

Task switching can consume resources that would be better used otherwise. A lot of waste is created when switching back and forth between different tasks. Especially when developers switch tasks they must familiarize themselves with the new task to be able to work on it efficiently. Depending on the task it can take from a few hours to a couple of weeks.

Waiting is considered wasted time especially when waiting for instructions or information that is required to continue working. In companies that have different branches working together on project waiting is a normal occurrence. Several customers or several outside companies that are working together on a project can also lead to a lot of waiting. Especially in cases where a developer has a question it can take a long time for that question to be answered if it goes through a chain of people before going to the people having the answer. While waiting, the developer does other tasks and this transition takes time as described in *task switching* that could have been spent on the original task had the question been answered fast.

Motion that is required from the developer is time that could be spent more productively.

It can be time that is required to walk to a co-worker to ask a question or get help for a technical problem. When artifacts or tasks are moving, it is especially wasteful. Handoffs of tasks create a lot of waste as all knowledge cannot possibly be included in the documentation. Handoffs can be minor, such as switching a task from one developer to another or major such as switching a project to another team completely. Knowledge is lost when a developer is switching from one task to another or even from one project to another. The project that lost experts will require support for some time after the handoff which is can be considered as wasted resources.

Defects are lost value as something has been developed and it is not working perfectly and must be fixed. After finding a defect follows time consuming testing, development and verification in addressing it. If a product is released with a defect in the system, it could become very costly for the company.

The heart of lean development consists of an iterative development model which is vital to removing most of the waste in an organization. After waste has been removed the lean principles can help in promoting a continuous drive for added value and improved processes.

Amplify Learning

It is next to impossible to have the full knowledge of the system at the start of a project in any mildly complex software venture. It can cause problems with the classic software design plans such as Waterfall, where large parts of the system are planned out before any code is written. In case of unexpected technical issues or requirement changes the system will have to be redesigned which creates problems. Lean Software Development methods solve this issue by using short, full-cycle iterations. Each iteration leads to working, tested and deployable code that is based on the customer’s requirements. Changing requirements and technical issues are better handled with this method than with the classic software development methods.

Feedback to and from the customer is one major action point in Lean development. By communicating with the customer about requirements and possible problems, lean developers increase their own and the customer’s knowledge of the system can better provide a product that meets the needs of the customer. Constant communication reduces the chance of creating extra functions that are not actually needed by the customer, which in turn reduces overall waste. As the customer is involved in the decision process they have a better understanding of the system being built and they can make informed requests for the development team.

Decide as late as possible

Changing requirements in the middle of the development process is so common in software development that it is almost an expectation for any project. It is a problem for fully specified projects and to minimize the waste of unexpected changes Lean developers will delay committing to a decision until the last possible moment. Delaying decisions allows developers to better research a problem which gives them better ability to make a decision than at the start of a project. Delay on decisions also keeps the options open giving developers time to handle the uncertainties, even those created by an indecisive customer. End goal of a system is the possibility that any new feature can be added at any time, which is accomplished by removing dependencies and leaving options open is a way of starting that.

Deliver as fast as possible

As Lean software development uses the feedback model, where communication with the customer is constant and decisions on requests are made as late as possible, the results from those requests should be done smoothly in one flow of events. In lean development, the flow of development is optimized to minimize the time from request approval to deployment it is possible and expected to deliver quality products in short time periods.

Fast delivery offers benefits for customer requests. In older development models customers can change their minds on requests in the middle of the project, while in lean development they don’t have the time to do that. Requests are quickly approved and then implemented usually in the next iteration. Customers can see how their first request works and then they can make more knowledgeable requests in the future.

Delivering as fast as possible requires keeping tight control of each developer’s work-in progress limits. Developer should only have such amount of work that can be consistently handled for the next incremental release. All benefits of fast delivery are lost if work begins to pile up with no time to handle it. Any partially completed work is subject to becoming outdated or worthless, i.e. waste. Fast delivery on requests is the only way to reliably keep meeting the expectations on those requests.

Empower the team

Delaying decisions is important but also is making the decisions as close to the people doing development, preferably by the developers themselves. Iterative model is only useful if the developer who is writing the code can make quick decisions on how to create the product the customer has requested. Strict documentation and models are used in the older development models, but it is beneficial if the documentation is more like guidelines and goals for the specific work item.

By allowing the developers to make the decisions themselves based on the guidelines and goals, they can better make informed decisions for their work when problems arise. Short development cycles and constant feedback with the customer allows the developers to quickly see the results of their decisions and can then continually improve the decision process. Broader goals of the project are handled by managers and they don’t have to focus on tiny details of the implementation themselves.

For this low-level decision-making process to work, the developers must be knowledgeable and experienced in their work and be able to quickly access necessary information. If that is not the case, allowing the team to make decisions on things they are not qualified for will create more errors and wasted features, which defeats the benefits of low-level decision making. Communication and feedback between the customer and the developer is the most important source of relevant information for critical design issues.

Build integrity in

Per Marry Poppendieck, integrity in software is one of two varieties: perceived integrity and conceptual integrity. Both add value to the customer and eliminate waste which gives the developers ample reason to add them to the product they are building.

Perceived integrity is something that gives the customer exactly what they want, even if they don’t themselves realize it. It is only achieved through continuous communication with the customers. Incremental releases allow the customers to see how their requests have been implemented which gives them the possibility to give more informed requests for future improvements.

Conceptual integrity in a product means that the subsystems of a system can interact with the work flow and work together flawlessly. It is achieved through continuous communication between developers and customers about the individual parts of the product and how the customers view the entire software. As developers communicate with other developers working on other parts of the product they can plan on how to make their individual decisions align with the other developers. This will allow the whole product to work cohesively even as its parts are made by different developers.

See the whole

Focus on short iterations and feedback model can produce valuable parts of the product quickly, but the entire project needs to be considered as well. Optimizing the whole project instead of optimized parts is important in order to gain a complete and well optimized product. Developers must communicate with each other to successfully integrate different parts to a cohesive product.

Developers don’t have to worry about the projects scope because of the cycle of small increments. As the projects scope can change at any time, it becomes an obsolete measurement. Lean development focuses on meeting the customers wishes in every cycle while keeping in mind the high-level domain requirements. By concentrating on these two principles in developing the product, the scope will align with what the customer truly wants. (Poppendieck and Poppendieck, 2003)

Advantages of Lean Software Development

1. Improved efficiency for the development process is the product of the elimination of waste. It speeds up the development and reduces project time and cost.
2. Lean Software Development also improves the delivery time of the product which in turn leads to more delivered products.
3. Empowering the developers to make their own decision gives them greater motivation for their work. This increases the product quality as the developers are more invested in

the development process as they would be in a development process using standard methods such as waterfall.

Disadvantages of Lean Software Development

1. Heavy dependency on team cohesiveness can become a problem if the team isn’t well assembled. Team composition regarding their skills and expertise is vital to the success of Lean Software Development and it can cause issues if the team doesn’t possess the required skills.
2. Decisions by the customer can become a hindrance if they are not promptly delivered to the development team on important decisions as it will slow the development process.
3. If business and software requirements evolve too much and the scope is not kept a handle on, the original objective might get lost and will never get finished.

Kanban

Kanban is a methodology that is used for definition, management and improvement of services that produce knowledge work, for example the design of software products. It is based on giving visibility to otherwise unseen knowledge work, and to ensure that the service being delivered works on the right amount of work. Only work that is requested by the customer and actually needed will be produced. Kanban system is a delivery flow system that is aimed at limiting the amount of work in progress (WIP) by using visualized signals.

Kanban is focused on the delivery of services by people working together to produce work products. A service has a customer, who is the one making the requests for work and who accepts or acknowledges delivery of completed work. Value is mostly produced as informational content such as software even when there are physical products from the services.

Kanban board contains the work flow of items through various stages of a process. It has signals to limit the work in progress and commitment and delivery points in the flow as shown in the figure below. Items taken into Analysis are committed to developing and items in Deploy are delivered to the customer. WIP limits are shown as the numbers above the different stages. The WIP limits create a pull system where work is only “pulled” into the system where there is capacity available. Work is never "pushed" into it when the capacity is already at maximum to prevent bottlenecks.

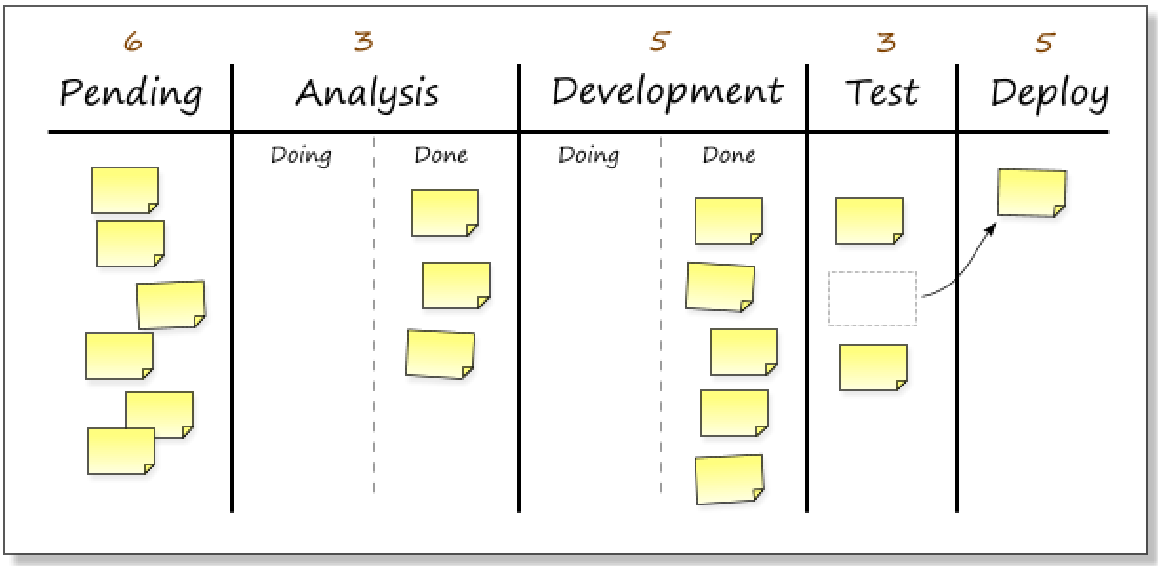


Figure 4. An example of a kanban board (Peterson, 2015)

There are six practices that define the essential tasks for person managing a Kanban system which were presented by Anderson and Carmichael. (2016)
They are:

1. Visualize
2. Limit work in progress.
3. Manage flow.
4. Make policies explicit
5. Implement feedback loops
6. Improve collaboratively, evolve experimentally.

Visualize: Kanban board is used to visualize work and process it goes through. It needs to contain commitment and delivery points and WIP limits in order to be a Kanban system. The visualization of work and policies is an important part of collaboration and gaining understanding of the current system which allows for finding of potential improvement items.

Limit Work in Progress: WIP limits change a “push” system to a “pull” system where new work is not started before current work is finished. Having too much un-completed work is wasteful, and it lengthens lead times which prevents the organization from responding fast to their customers or circumstances. Optimizing the amount of work in progress is vital to the success with Kanban. It leads to improved lead times for services, higher quality and faster deliveries.

Manage flow: In a Kanban system, the flow of work should maximize the delivery of value, minimize lead times and be as predictable and smooth as possible. Managing the flow is important as there are usually conflicting goals during the process. Things such as bottlenecks and blockers have to be identified and managed in order the achieve the best possible flow of work. Cost of delay of work items is the key in understanding and maximizing the flow of value. It is the amount of item’s value that is lost by delaying its production by a specified period. Four archetypes are used to characterize how the value of items changes with delay which are: expedite, fixed date, standard and intangible. They can be used to assist in ordering work items through the work flow.

Make policies explicit: Policies for the people using the Kanban system needs to be set explicitly. When everyone is working with the same rules, the process flow goes smoothly and emerging problems are more easily noticed and acted on. Example of explicit policy can be definition of done. Before certain criteria is met, an item cannot be taken forward in the work flow.

Implement feedback loops: Continuous feedback and improvement is encouraged by Kanban. It has defined seven specific feedback opportunities called cadences. They are cyclical meetings and reviews that strive towards evolutionary change and more effective service delivery. It is not required to add seven new meetings to organizations overhead, instead they can be included in existing meetings.

Improve collaboratively, evolve experimentally: Kanban is at its core an improvement method. When starting it assumes the current state and using the flow system in seeing work as a flow of value in pursuing continuous and incremental improvements. Such process has no endpoint as perfection in the ever-changing environment is unreachable. Beneficial change is vital to an organization, not changing and evolving leads to extinction. (Anderson and Carmichael, 2016)

2.3 Scaled Agile Framework

Introduction to SAFe

The Scaled Agile Framework for Lean Enterprises is a framework composed of Agile and Lean practices that is designed to assist organizations in delivering new products and services in shorter timeframes with best quality and value possible. It is scalable from smaller organizations to big ones consisting of thousands of people.

SAFe is based on Lean-Agile principles and values to form an extensive body of knowledge. It provides instructions for the roles, responsibilities, artifacts and activities that are necessary to achieve greater business value. It combines Agile and Lean product development with systems thinking while synchronizing alignment, collaboration and delivery for multiple Agile teams. This results in improved business agility by accelerating productivity, time-to-market, quality, employee satisfaction and more.

SAFe 4.5, which is the latest version contains four levels which are from top-down: Portfolio level, Value Stream level, Program level and Team level. Value Stream level is not used in the company where the research is conducted so that level is omitted from presentation. Portfolio, Program and Team levels will be presented in more detail later.

SAFe Core Values

Core Values in SAFe define the ideals and beliefs that are essential to successfully using the framework. They help people in knowing where to focus their efforts and assist organizations in determining if they are on the right path to fulfilling their business goals. The Core Values are *Alignment, Built-in quality, Transparency* and *Program execution.*

1. *Alignment* – Focus from the complete portfolio - management and teams - is required to reach alignment to a common mission. When that is achieved all the energy is directed towards helping the customer reach their goals for the product. Alignment

communicates the goal for the project and allows team to focus on how to accomplish it. Alignment is reached when everyone in the portfolio, including every team member on every Agile Release Train (ART), is aware and understands the strategy and their role in accomplishing it.

2. *Built-in quality* – Poor quality creates a much higher economic impact at higher scale. Built-in quality practices are designed to increase customer satisfaction and to allow for faster and more predictable deliveries. They also increase the innovation capabilities and allow for risk taking. Without built-in quality, the Lean goal of maximum value in the shortest sustainable lead time is unattainable. Its practices also ensure that appropriate quality standards are achieved for each solution item, during every increment.
3. *Transparency* – Transparency promotes trust among co-workers. Trust is essential for performance, innovation, risk-taking and relentless improvement. In large-scale solution development, things don't always go as planned, which makes it important to create an environment where telling facts is always welcomed in order to build trust and increase performance. This kind of environment enables fast and decentralized decision-making and increases employee empowerment and engagement.
4. *Program execution* – Entire development value stream must be lean and responsive to change in order to achieve broad change. Traditional organizational structures and practices were built to achieve control and stability, and they were not designed to support innovation, speed and agility. SAFe deliver value by creating stable teams-of-Agile-teams that form an Agile Release Train. It is a self-organizing and self-managing organizational structure that delivers value to the customer. ART relies on decentralized decision-making that helps it avoid delays that are caused by having to wait for information and decisions to travel up the chain of command.

The Lean-Agile Mindset

The Lean-Agile Mindset is a combination of beliefs, assumptions and actions that are the basis of way of working for SAFe leaders. It is the foundation for applying SAFe principles and practices and teaching them to others.

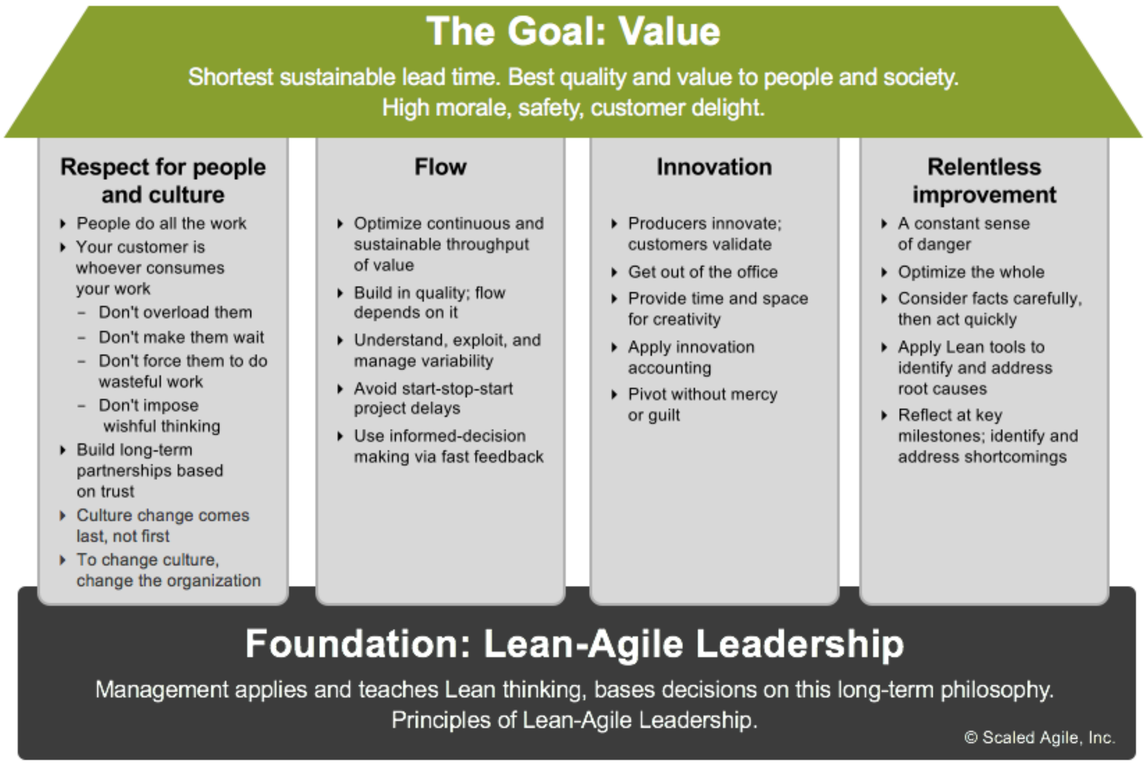
SAFe is based on Agile development, systems thinking and Lean product development. Agile gives the tools and practices required to empower and engage teams to reach increased levels of productivity, quality of releases and engagement. In order to support Lean and Agile development in bigger scale across an entire enterprise, a deeper and broader Lean-Agile mindset is required. There are two primary aspects of a Lean-Agile mindset:

- Thinking Lean – SAFe House of Lean represents most of the Lean thinking. The roof represents the goal of delivering value to the client. The four pillars stand for respect for people and culture, flow, innovation and relentless innovation. Lean leadership provides the foundation for the House of Lean.
- Embracing agility – Skills, ability to learn and abilities of Agile teams and their leaders is what SAFe is built on. The Agile Manifesto provides a system and principles that are essential to forming the mindset for successful Agile development.

These two aspects help create the Lean-Agile mindset as part of a new management approach and an improved company culture. It gives the leadership that is needed to fulfill a successful SAFe transformation which helps individuals and organizations reach their goals.

The SAFe House of Lean

Principles of Lean, derived originally from Lean manufacturing and then applied to software, product and systems development are now widely in use. Poppendieck's Lean thinking aspects and principles have helped create the SAFe House of Lean presented in the picture below.



The Goal: Value
Shortest sustainable lead time. Best quality and value to people and society.
High morale, safety, customer delight.
Respect for people and culture
People do all the work
Your customer is whoever consumes your work
Don't overload them
Don't make them wait
Don't force them to do wasteful work
Don't impose wishful thinking
Build long-term partnerships based on trust
Culture change comes last, not first
To change culture, change the organization
Flow
Optimize continuous and sustainable throughput of value
Build in quality; flow depends on it
Understand, exploit, and manage variability
Avoid start-stop-start project delays
Use informed-decision making via fast feedback
Innovation
Producers innovate; customers validate
Get out of the office
Provide time and space for creativity
Apply innovation accounting
Pivot without mercy or guilt
Relentless improvement
A constant sense of danger
Optimize the whole
Consider facts carefully, then act quickly
Apply Lean tools to identify and address root causes
Reflect at key milestones; identify and address shortcomings
Foundation: Lean-Agile Leadership
Management applies and teaches Lean thinking, bases decisions on this long-term philosophy.
Principles of Lean-Agile Leadership.
© Scaled Agile, Inc.

Figure 5. The SAFe House of Lean (Scaled Agile, 2015)

Delivering maximum customer value in the shortest sustainable lead time while providing the highest possible quality to customers and society, is the goal of Lean. The house of Lean contains four pillars which together help in reaching this goal.

Pillar 1 – Respect for People and Culture: People create the products and value to the customer. Creating a culture of respect gives employees a sense of belonging and empowers them to create better products.

Pillar 2 – Flow: Successful execution of SAFe requires a continuous flow of work which supports incremental value delivery.

Pillar 3 – Innovation: Product and processes required innovation of they will steadily decline. Innovation is critical part of the House of Lean, as it not only stops the decline but enables the creation of even better products.

Pillar 4 – Relentless Improvement: Continuous reflection and feedback on processes enables improvement which is necessary so that the company doesn't stagnate. Always improving and learning keeps the organization in a good place to continue to keep providing value to their customers.

Foundation – Leadership: Leaders create the foundation for the successful adoption of Lean-Agile approach. Without engaged leadership adoption of new ways of working is highly unlikely.

Levels of SAFe

Portfolio level

Portfolio level is the highest level in SAFe. It contains the governance elements for organizing the Lean-Agile Enterprise into one or more value streams which develop solutions that produce the business value. Program Portfolio Management represents the stakeholders who are accountable to deliver the business results. One of the primary responsibilities of the program portfolio management is the allocation of funds to the value streams. At the portfolio level Business Epic and Epic enablers are discovered, defined and administered. Business Epics capture and reflect the new business capabilities that can only be provided by cooperation among value streams. Epic Enablers consist of architectural and technical initiatives that are required in order to enable the development of the new capabilities.

Portfolio Kanban system is used to manage the flow of work and to make sure it visible to all stakeholders. The Kanban system gives the work visibility and provides work-in-process limits to help assure that demand is metered to the actual value stream capacities. Portfolio backlog is the highest-level backlog in SAFe, and it serves a holding stage for epics that make it through the Kanban systems to await implementation. Epics in portfolio backlog are prioritized accordingly. Epic Owners take the responsibility for managing epics at minimum until implementation is assured.

Program level

At the program level development teams and other resources are organized to an Agile Release Train (ART) that deliver solutions. ARTs are a team of teams consisting of 5-12 teams that plan, commit and execute together. They deliver value in Program Increments

(PIs) that are 10 weeks in duration. Each PI consists of 5 2-week iterations (Sprints). The Agile Release Train is aligned to a common mission via a single Program Backlog containing the development items for the train. Features and Enablers are maintained and prioritized in the program backlog to ensure visibility to stakeholders and manage flow of work. Features are sized to fit in a program increment (PI) so that each PI delivers new functionality. Features are split into stories that are developed by a single team within an iteration.

Team level

Team level contains the organization, artifacts, roles and process models for the activities of an agile team. Agile team is responsible for defining, developing and testing stories from their Team Backlog. Development is done in fixed length iterations and synchronized with other agile teams in order to ensure that the entire system is iterating. Teams use ScrumXp and Team Kanban with Built-in Quality practices to ensure the quality of the product. A single team consists of five to nine members. The whole environment in enterprise scale in which a team works is seen in the picture below in the lowest section from the bottom. It visualizes that one team is but a one piece in large puzzle that is the ART.

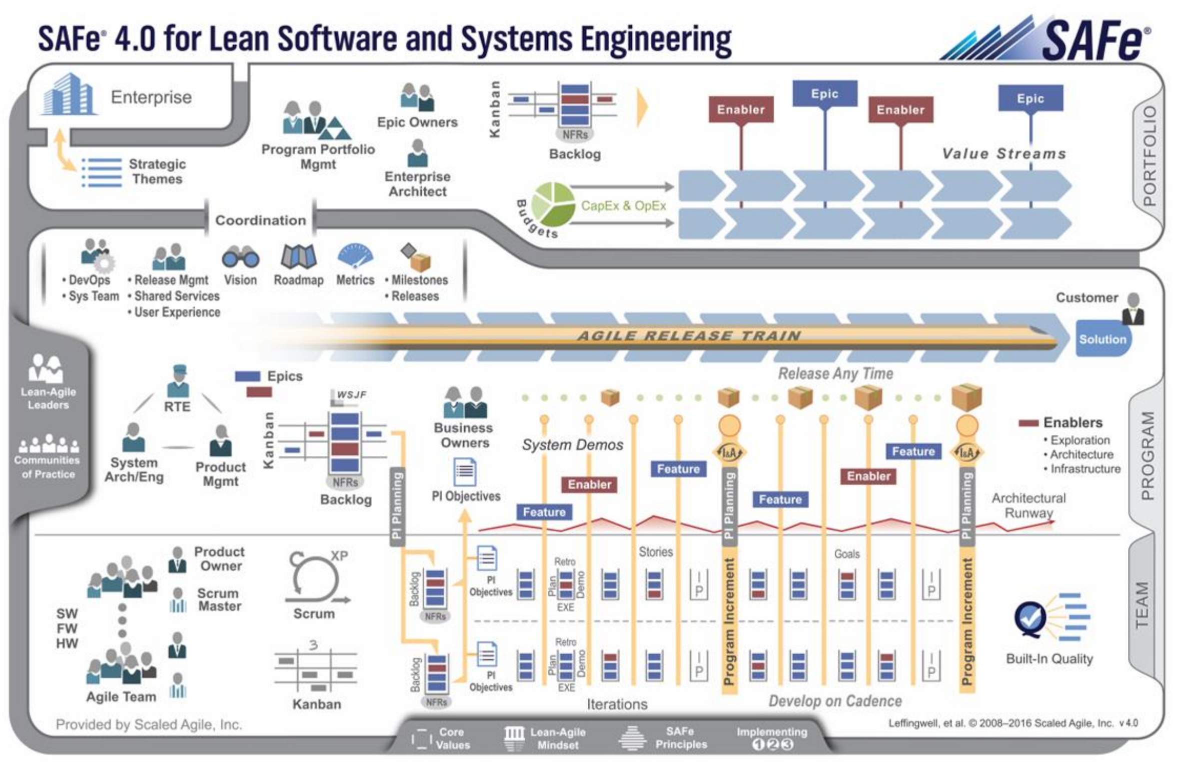


Figure 6. SAFe 4.0 (Inno.com, 2016)

SAFe roles and organizational governance

Program Portfolio Management represents the people with the highest-level decision-making responsibility concerning the strategy in the portfolio. They are the stakeholders accountable for delivering business results. Program Management is the highest governance function within the framework. They guide the day to day work at portfolio level and remove impediments while facilitating decision making. The role directly corresponds the role of a Release Train Engineer at the program level.

Solution Portfolio Management is responsible for the content that is created in the multiple Agile Release Trains. They make sure the Epics being produced are aligned with the strategic themes of the portfolio. It corresponds the role of Product Management at the program level of the framework.

Enterprise Architect is responsible for creating the architectural strategy – and roadmap for the portfolio and that it is followed. Overseeing the lifecycle of the information systems being used is also a responsibility for the Enterprise Architect. It corresponds the role of a System Architect at the program level.

At the team level the roles responsible for content and governance are Product Owner and Scrum Master. They make sure that the vision from the portfolio level regarding to content is being produced by the team and that they work as productively as possible. Agile team looks to guidance from the System Architect on matters related to the architectural strategy and tools.

SAFe ceremonies

A SAFe Agile Release Train contains multiple ceremonies aimed at keeping the different parts of the train in sync and going in the same direction. Ceremonies at the program level are PI Planning, Scrum of Scums, Program Owner Sync, Release Management meeting, System demo and Inspect & Adapt. PI Planning is where the work of the train for the next PI is planned and all agile teams are aligned toward a common goal. It is the most important ceremony in an Agile Release Train and everyone from the train will be attending the event. Scrum of Scrums is an event where a representative from each Agile team gives a consolidated update for their respective team. Open train-wide impediments are discussed along with open business- and technical queries. Product Owner sync is organized weekly to gain visibility into how well the ART is progressing toward its PI objectives. Problems or opportunities with Feature development is also discussed during this meeting. Release Management meetings provide governance for upcoming releases while also providing regular communication to management. System demo is a bi-weekly/weekly event that provides the customer a view of new features produced in the ART by the different teams. Inspect & Adapt is an event where the current state of the product is demoed and evaluated. Teams in the train reflect on their work during the last PI and identify improvement items that can be added to the backlog for future PIs.

Team level ceremonies include Daily Stand-up (DSU), Backlog Refinement, Team Demo, Iteration Retro and Iteration Planning. DSU is a daily meeting of about 15 minutes where the team discusses progress and impediments. Each member gives an answer to three questions: “What was achieved yesterday?”, “What is planned to be done today?” and “What are impediments?”. Impediments should be discussed and addressed at the end of the meeting. Backlog Refinement’s purpose is to preview and elaborate on upcoming stories. Team gets an idea about coming work before the Iteration Planning. It provides the team with time and context to identify possible dependencies that could impact the next iteration. Team Demo is used to give a demo of completed stories to the team and Product Owner. Iteration Retro happens at the end of each sprint, where the entire team meets to assess what went well and what can be further improved. Scrum master keeps track of possible action items from this meeting. Is usually kept as a combined meeting with Iteration Planning. At Iteration Planning the organization of work and definition of a realistic scope is done for the next iteration. The team agrees on a set of stories for the upcoming iteration and summarizes those for a set of iteration goals which are based on the team’s capacity.

Backlog management

Backlogs contain the work items in SAFe. Presented in the picture below is how the higher-level work items (Epics) are split up to features for Agile Release Train and then to Stories for Agile teams.

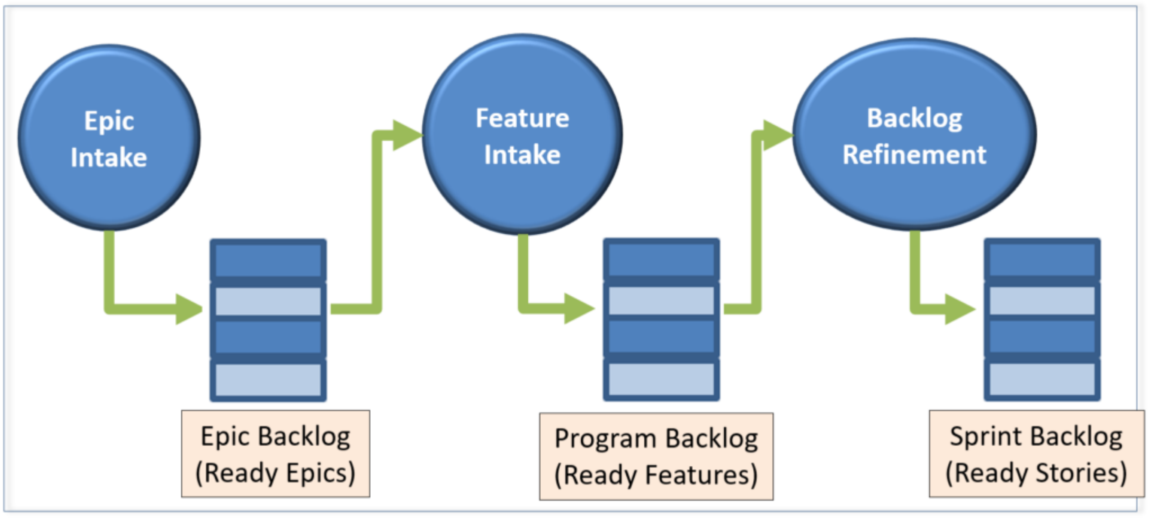

Epic Intake
Feature Intake
Backlog Refinement
Epic Backlog (Ready Epics)
Program Backlog (Ready Features)
Sprint Backlog (Ready Stories)

Figure 7. Interconnected Backlogs (The Burndown, 2018)

Portfolio backlog

Portfolio backlog is the highest-level backlog in SAFe. It holds the upcoming Business and Enabler Epics required to create solutions that address the strategic themes and facilitate business success. Epics in the portfolio backlog have been reviewed, analyzed and approved for implementation on their way through the Portfolio Kanban system. Estimation of the Epics is done in the analysis step. Estimating gives the enterprise a sense of the future, which is required to support effective planning and execution.

Portfolio Backlog is owned and prioritized by the Program Portfolio Management. Epics in the portfolio backlog are ranked relative to each other, usually by WSJF (weighted shortest job first) prioritization. Business and Enabler epics are typically compared only to each other under the limits of capacity allocation for each type. Highest ranked epics will be ready for implementation and they are pulled from the backlog when ARTs have available capacity to implement them. Epics ready for implementation are split into program epics which are moved to ARTs program backlogs where they are further split into features.

Program backlog

Program backlog consists of upcoming Features that are intended to address user needs and deliver business benefits. Enablers that advance learning and build the Architectural Runway are also present in the program backlog. Product Management manages the backlog by identifying, refining, prioritizing and sequencing the items in the backlog. Program Backlog is prioritized using the WSFJ prioritization. Developing, maintaining and prioritizing of the backlog is done constantly by the Product management in preparation for the next PI. A single ART has a single Program Backlog, which only contains work for that train.
Features that are planned and have achieved Definition of Ready can be taken to the planning of the next PI where they are broken down into Stories at the PI planning and allocated to agile teams.

Team backlog

Team backlog contains items a team needs to do to advance their progress toward building the system. It can contain user or enabler stories, which are mostly originating from the program backlog. It can also contain tasks that are relative only to the team's specific context. Team backlog contains all the work that the team is expected to complete during the PI. Team backlog is owned by the Product Owner and he/she prioritizes the items in the backlog for execution. Stories that have achieved Definition of Ready can be taken into Iteration planning. (Scaled Agile, 2018)

2.4 Team performance

A team consists of a small number of people working toward a common goal. They possess skills that complement each other, and they are committed, have set performance goals and approach for which they hold themselves mutually accountable. Team also has common tasks; its members interact with each other and they work in the same organizational context. Five factors have been identified through systematic review of empirical studies that influence to software development teams' performance. Those factors are: team coordination, goal orientation, team cohesion, shared mental models and team learning. They will be presented in more detail below. (Dingsøyr et. al, 2016)

Team Coordination

Development of software is not a straight up planned activity where nothing changes during the development time. Plans constantly change which makes the teams' ability to adapt to changes extremely important. As requirements and technologies are subject to changes, coordinating team members is also important for project success and achieving quality.

Coordination in software development context means managing dependencies between activities. Dependencies include items such as shared resources, task assignments, and task and sub-task relationships. In order to manage dependencies successfully feedback is required as teams are not always able to identify dependencies before they start work on a task. Teams' performance is increased by frequent feedback.

Team coordination involves establishing a common understanding in the team regarding dependencies. This includes coordinating work processes, creating internal procedures and mechanisms for feedback and coordinating team members' contributions. Aligning individual contributions required mechanisms for coordination, like common work breakdown structures, schedules, budgets and deliverables.

Member interaction is vital to team coordination. Project outcomes improve with better and more frequent interaction between team members. Managers use higher level project plans to assign work, they allocate physical and economical resources, manage dependencies between resources, and integrate outputs. Coordination tools for administrative work include budgets, staffing tables, critical-path analysis, milestones, inspections and review meetings. Overall performance is increased by usage of these tools. For software teams, the methods dictating development also determine the most effective coordination mechanisms, for example how to plan work and allocating resources to different tasks. Continuous communication and feedback on work items and coding standards also increases team performance.

Goal Orientation

As mentioned before, a team has a common purpose that it works towards and a set of performance goals. Performance is affected by achievement orientation, a goal-oriented team leader and by the teams' ability to define clear goals for themselves.

Establishing clear and long-term goals and milestones while planning is action teams should take. Software development is complex, so it is important for team members to also have mid-range goals to keep them on track and increase performance. For a team leader, goal orientation consists of actions. It can be considered a personality trait, meaning a person has strong inclination to set, reach for, and achieve goals. A goal-oriented leader greatly increased individual and team performance, including schedule and resource management. In development teams, the leader is usually a project manager who is responsible for project plans and communication with stakeholders.

Team leadership is focused on influencing the team members to work towards mutually agreed project goals. Good leaders understand the development process's dynamics and by monitoring and evaluating team members behavior are able to lead the team towards its goal. Evaluating the results of team members' work is an alternative way to leading the team. Performance is better if everyone, not just the team leader evaluates the teams' results.

Some development approaches, such as agile, promote self-managing teams. Few studies however show that self-management impacts performance. Regardless of development approach, goal orientation is important for team performance.

Team Cohesion

Team cohesion, as described by Mudrack is “the tendency for a group to stick together and remain united in the pursuit of its goals and objectives.” (1989). It primarily includes commitment to team tasks but also contains team wide personal attractions and chemistry among team members and group pride.

A study researching the influence of team cohesiveness, experience and capabilities on performance found out that cohesiveness was the most important (Kang et al, 2006). Also, another study found that cohesion was the most important factor in linking quality of teamwork to performance (Dingsøyr and Dybå, 2012).

Extreme Programming, among other agile-development methods advocates collective ownership of code. It leads to fewer bugs which is a measure of software quality. Developers working on code done by other developers is a form of intensive collaboration, which is not likely to be successful in a team that is not highly cohesive. This is one example of how team cohesiveness can have an indirect impact on the quality of products.

Teams with conflicts are the opposite of cohesive teams. Conflicts can negatively impact performance, product success and customer satisfaction. Different types of conflicts affect differently. Relationship conflicts hurt performance, while task-related conflicts can enhance it. This can happen because task-related conflicts give the team new ideas and makes it avoid groupthink.

Conflict management is important as conflicts are inevitable in teamwork. A study measuring team and product performance found that teams that focus on conflict management do better than teams that don’t (Gebert et al, 2006). There are several ways in which to manage conflicts. Recognizing disagreements over a solution and working together to solve the disagreements boost performance as opposed to imposing a solution on the team causing issues.

Shared Mental Models

Team members’ capability to acquire, communicate and apply relevant knowledge is vital in software development. Shared mental models represents the knowledge and skills team members' have in common that allows them to understand tasks and relationships and dependencies between them. They are able to coordinate their actions to the good of the team. Scrum is an example of a shared mental model, as in it the team shares a common understanding of the project's main goals and how they are related to each other.

A team which has established a shared mental model has members who can anticipate each other's needs and can thus adjust work strategies more easily if there are changes with tasks. Shared mental models can make it easier to understand and explain how team members work together. As with the other disciplines it has been found that shared mental models improve team performance.

Knowledge sharing and communication among team members leads to shared mental models. The models can include a common understanding of the team's goals which can also improve performance. Expertise and knowledge among the team members is an important factor as fewer shared mental models among the team increases a product's time to market.

A study comparing the effects of mental models with the effects of member demographic similarities such as age and gender, found that shared mental models have a more positive impact on the outcomes (Kang et al, 2006).

Team Learning

Team learning increases skills and expertise of team members and improves performance, effectiveness and job satisfaction. For it to be useful it must be an ongoing process where people use reflection and action, ask questions and seek feedback, experiment with different solutions and discuss results and errors. These actions allow the team to become reflexive which means they can adapt to changing circumstances by adjusting their objectives, strategies and processes accordingly. Team learning increases the teams collective level of knowledge and expertise. With their increased knowledge they can better make the decisions on whether to adapt or improve.

For software development teams, skills and performance directly affect performance. Different skills, such as those related directly to completing variety of tasks, development methods and application domains, increase performance. Based on research, it's been found that expertise coordination among the team is more important for performance than its members' seniority and project plans (Malone and Crowston, 1994).

Comparison to Agile-Development Principles

Comparing the five propositions from earlier with the Agile Manifesto's 12 principles give the following conclusions.

Team Coordination – Agile principle states that software should be delivered frequently in short iterations. Short iterations require good team coordination. Principles don't explain how to coordinate but in one of the agile methods – Scrum – coordination is done through short meeting (daily stand up) as well as common ceremonies including iteration planning, review and retrospective.

Goal-Oriented Leadership – Based on agile principles the best architectures, requirements and designs come from self-organized teams. Also, one principle states that "working software is the primary measure of progress". Agile methods however do not offer guidance on how the self-management approach can help reach project goals better. They don't give clear picture on whether self-management leads to goal orientation.

Team Cohesion – One agile principle states that business owners and developers must work together daily during the entire project time. Scrum includes meetings that emphasize cohesion such as the daily stand up meetings as well as iteration planning and review meetings. Extreme Programming has even more emphasis on close cohesion as its practices include pair programming and shared code ownership. Team conflicts aren't discussed in Agile methods other than giving places where decisions are made, and conflicts can be negotiated. Agile principles only offer little advice about cohesion while concrete practices support it.

Shared Mental Models – Shared mental models aren't strictly mentioned in Agile principles, but their importance is described: "The most efficient and effective method of conveying information to and within a development team is face-to-face conversation". Even if they aren't explicitly mentioned, Agile methods themselves are strong shared mental models. They mostly have a few roles with a set of practices and artifacts which can make it easier to reach common understanding on how to do the development as the methods are relatively simple.

Team Learning – Regular learning and its importance is mentioned in one Agile principle. Team should reflect on how to become more effective at regular intervals and then adjust accordingly. This practice is focused on process improvement via retrospectives held at the end of each iteration. Other agile practices emphasize learning about domains and technologies, such as demos of new functionalities kept regularly and pair programming. (Dingsøyr et al, 2016)

Components of team work

Teams require an intricate mix of factors including organizational support, individual and teamwork skills. "Big Five" framework of teamwork by Salas et al. has five components of teamwork which are: team leadership, mutual performance monitoring, backup behavior, adaptability and team orientation. All five components are requirements for team performance. They need three coordinating mechanisms to function together: shared mental models, closed-loop communication and mutual trust. Each component of the framework is described in detail in the following table.

Teamwork component	*Definition*
Team leadership	Ability to direct and coordinate the activities of other team members, assess team performance, assign tasks, develop team knowledge, skills and abilities, motivate team members, plan and organize, and establish a positive atmosphere
Mutual performance monitoring	The ability to develop common understandings of the team environment and apply appropriate task strategies to accurately monitor team-mate performance

Backup behavior	Ability to anticipate other team members' needs through accurate knowledge about their responsibilities. This includes the ability to shift workload among members to achieve balance during high periods of workload or pressure
Adaptability	Ability to adjust strategies based on information gathered from the environment through the use of backup behavior and reallocation of intrateam resources. Altering a course of action or team repertoire in response to changing conditions (internal or external
Team orientation	Propensity to take others' behavior into account during group interaction and the belief in the importance of team goal's over individual members' goals
Shared mental models	An organizing knowledge structure of the relationships among the task the team is engaged in and how the team members will interact
Mutual trust	The shared belief that team members will perform their roles and protect the interests of their team-mates
Closed-loop communication	The exchange of information between a sender and a receiver irrespective of the medium, where the information is received

Table 2. Components of the "Big Five" framework of teamwork (Salas et al, 2005)

A focus group study by Dingsøyr and Lindsjørn looking at how practitioners in agile software development teams perceive factors affecting team performance it was found the components of the "Big Five" fit well as the factors affecting team performance. The components formed three groups – consisting of components viewed to closely associate with each other – in the study 1) *team leadership* and *closed-loop communication*, 2) *shared mental models*, *team orientation*, and *mutual trust*, 3) *mutual performance monitoring*, *backup behavior* and *adaptability*. (Dingsøyr and Lindsjørn, 2013)

3 FACTORS AFFECTING TEAM PERFORMANCE

Research data was gathered from several Accenture SAFe Trains (ARTs) operating in one client by analyzing the results of Agile teams Sprint retrospective material. Every team filled in a questionnaire at the retrospective meeting by answering three questions: “What went well?”, “What didn’t?” and “What can we improve in?”. The answers were assigned to categories that are based on the literature assembled. Goal of the research is to identify what affects the performance of the Agile teams in a multi distance and everchanging development environment. Based on the results, proposals of possible improvement actions were introduced with focus on attempting to mitigate the most common issues.

3.1 Gathering of the data

The data for this research was compiled from collected Sprint retrospectives for Accenture Agile teams in one specific client. Teams included in the research were from SAFe-trains. All teams had all their members employed by Accenture. Total of six teams from four ARTs were identified as suitable for this research. As mentioned before in the literature section, one Program Increment in SAFe consists of five sprints. Sprints number 2 and 4 were chosen for gathering of the results. In case sprint results were not available for the sprints 2 or 4, then the next closest sprint was selected instead. To get as much data as possible all the available sprint retrospective data was looked at, regardless of when the team started to operate. Data was gathered from October 2015 to May 2018.

The research was compiled as a quantitative study attempting to identify items affecting the performance of Agile teams in a SAFe development environment. Data was collected by going through sprint retrospective materials for the chosen teams and gathering items which meet the following criteria:

- Has positive impact on team performance
- Has negative impact on team performance

Gathered data was then assigned to categories presented in the next chapter.

3.2 Chosen categories

The Sprint retrospective results were categorized to following main categories which are then split to subcategories. The categorization is based on the literature introduced in chapter 2 and on the personal experience of the researcher who has worked in one of the Agile teams under investigation.

Internal	**External**
Team coordination	Dependencies
Goal orientation	Administration
Team cohesion	Technical items
Shared mental models	
Team learning	

Table 3. Chosen categories split by main categories: Internal and External.

The main categories are *Internal* and *External*. *Internal*-category will contain all items found from the research data that are related to the Agile team's internal matters. In turn the *External*-category contains all items that are affecting the team from outside the team itself. All founds items are also split into positive and negative for each Main and subcategory.

Subcategories chosen for *Internal* items are team coordination, goal orientation, team cohesion, shared mental models and team learning. Team coordination subcategory consists of items related to coordination, dependencies and task allocation within the team. Goal orientation subcategory includes items related to setting goals for the team, for example sprint or PI goals. Team cohesion subcategory includes items that are related to team member interactions with each other and how they handle conflicts within the team. Shared mental models' subcategory consists of items relating to the teams common working processes such as SAFe ceremonies. Team learning subcategory includes items relating to teams' continuous improvement and learning or needing to learn new skills.

Subcategories chosen for *External* items are dependencies, administration and technical items. These subcategories were chosen based on the literature on the development method and how teams interact within it. Dependencies subcategory contains all found items related to dependencies. Those can be related to e.g another Agile team, key persons (team, contractors, client). Administration subcategory contains all items that affect the team with bureaucracy or items related to the used operating model (SAFe). Technical items subcategory consists of issues of technical nature such as test environment and test data issues. This subcategory was specifically chosen based on the personal observations of the researcher during more than two years working in an Agile team in the research scope.

4 RESULTS

A total of 85 sprint retrospectives data was gathered from the six Agile teams. Total of 366 items were identified as affecting the performance of the teams. Of those 366 items, 234 were categorized as *Internal* and 132 as External. The items were also split into positive and negative for a total sum of 180 positive and 186 negative items. These values can be seen in the table below. Items were assigned to the category that was closest to the generalized version of the specific item. For example. “certain persons had too much to do” would have been categorized to Team Coordination subcategory within the *Internal* main category.

Main category	**Items found**	**Items found %**	**Positive**	**Positive%**	**Negative**	**Negative %**
Internal	234	64 %	156	87 %	78	42 %
External	132	36 %	24	13 %	108	58 %
Total:	**366**	**100 %**	**180**	**100 %**	**186**	**100 %**

Table 4. Gathered data categorized to main categories including split to positive and negative items.

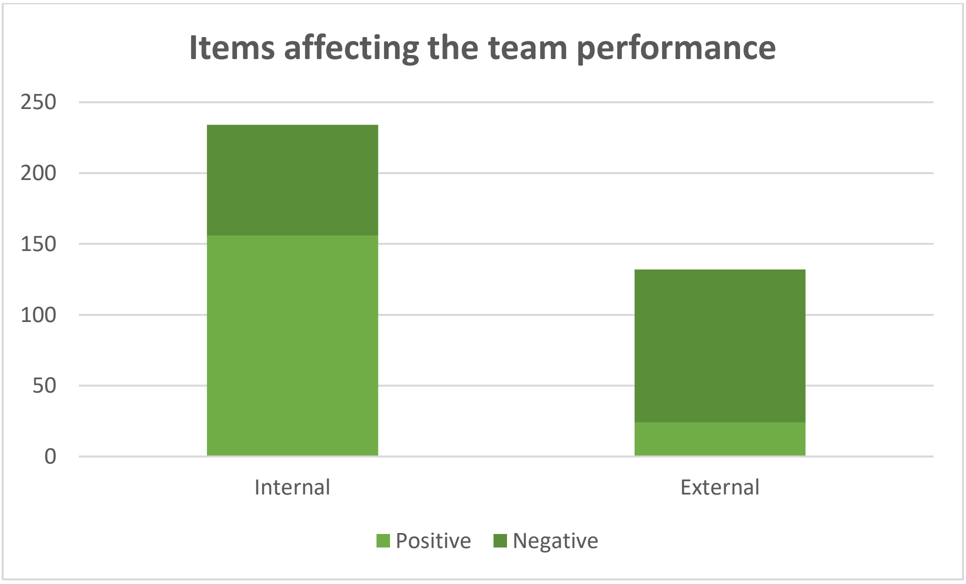


Figure 8. Found items split to Internal and External categories.

Majority of the items were categorized to the *Internal* main category but near even split was found between positive and negative items. Slight majority of the negative items were found in the *External* main category, even though it had nearly half of the total items in the *Internal* main category. It was to be expected that most of the found items would be categorized to the *internal* main category based on the research data used. It was expected based on the format of the sprint retrospectives which focuses on how the team can learn and improve from within.

58% of the negative items were found in the *external* category compared to 42% in the *internal* category. Teams are considerering that the items affecting their performance negatively are coming from outside of the team. 87% of the positive items were categorized to the *internal* category. So, according to the research it seems that teams consider items affecting their performance positively mostly to come from within the team itself. Issues which negatively impact performance come from outside the team and those are something the team cannot control.

Internal items affecting team performance

Items in the *internal* main category were split to subcategories according to the table below. Positive and negative split is also presented there.

Subcategory	Team Coordination	Goal Orientation	Team Cohesion	Shared mental models	Team Learning	**Total:**
Positive	42	57	15	9	33	**156**
Negative	31	20	0	4	23	**78**
Total:	**73**	**77**	**15**	**13**	**55**	

Table 5. Internal items categorized to subcategories with split to positive and negative items.

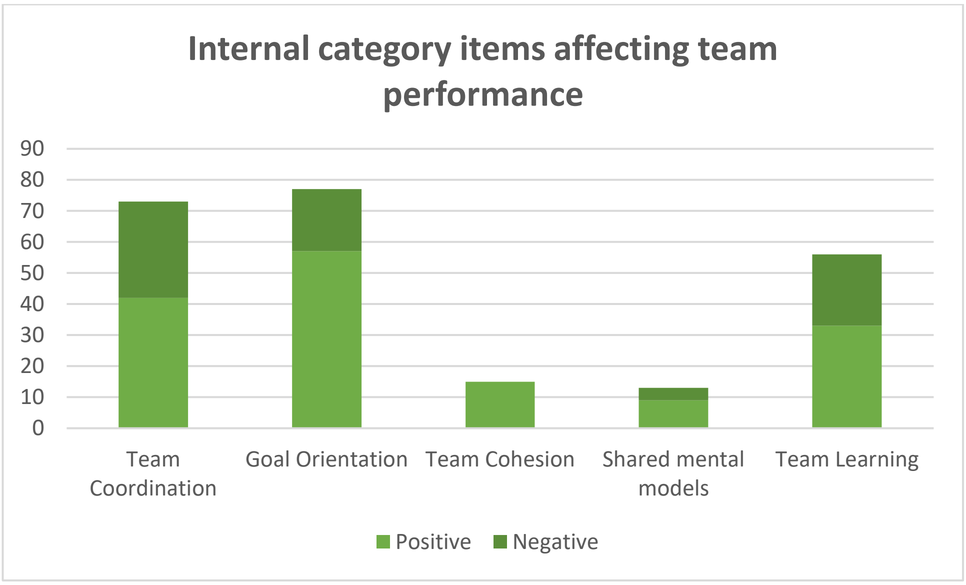


Figure 9. Internal items affecting team performance.

67% of the items categorized to the *internal* main category were considered to have a positive impact on the team's performance and 33% to have a negative impact. Most of the positive items found were either in the Goal Orientation or Team Coordination subcategories, with Team learning the third highest subcategory. Team Cohesion and Shared Mental Models subcategories had together only 15% of the positive items in the *internal* main category.

Goal Orientation subcategory had the biggest difference between positive and negative items with 74% positive items to 26% negatives. Positive example of answers in Goal Orientation category was "all tasks done". Example of negative item was "too much work in sprint". Teams consider that their sprint goals are most of the time well specified and they can be productive. According to the research poorly defined sprint planning scope and estimation of stories led to negative impact in team performance.

Team Coordination subcategory had 58% of the items considered to be positive with 42% items negative in turn. It can be said that teams consider that their internal coordination is at a good spot but that there is room for improvement. Different work load between team members was mentioned as both positive and negative item in the research data. Coordination within the team can differ greatly depending on if all team members are able to work from the same location or if a team has offshore team members who are working from different geographic locations. Mostly it is up to the team leader or scrum master to make sure everyone is kept up to date. Task assignments can be hard to solve as sometimes only certain people have the knowledge (e.g technical skills, functional knowledge) to complete some tasks. In ideal world everyone in an agile team is able to do everything but that will require lots of training internally and is often not possible.

Team Learning subcategory had 60% positive items and 40% negatives. Positive items were related to additional motivation when learning new skills or technologies. Negative effect to performance was when there was lack of skills to handle assigned tasks and additional training was needed.

Only 15 items were categorized to Team Cohesion subcategory and all were considered to have a positive impact on the team performance, for example “good team spirit” was one of the items categorized to this subcategory. Agile teams in this development environment seem to have good commitment to teams’ tasks and they can productively work together as a team. It should be noted however that in a gathering such as a sprint retrospective where all team members are present, negative items related to team cohesion might not always come up.

Shared Mental Models subcategory had the lowest number of items in the *internal* main category, only 6% of the total amount of answers. Of those 13 items in this subcategory 9 were positive and 4 negative. Low number of total items in Shared Mental Models subcategory and that 69% of those being positive points to that teams are having good common understanding on how they are working together and how to productively use agreed processes. Based on the results and low total amount of items found in this subcategory one conclusion can be that teams might not bring up items related to commonly used processes or the used operating model in the sprint retrospectives as they are too normal to them in the sense that they use the same processes every day.

External items affecting team performance

Items in *external* main category were split to subcategories in the following manner shown in the table below. Split to positive and negative items is also presented there.

Subcategory	Dependencies	Administration	Technical items
Positive	18	2	4
Negative	45	22	41
Total:	**63**	**24**	**45**

Table 6. External items categorized to subcategories with split to positive and negative items.

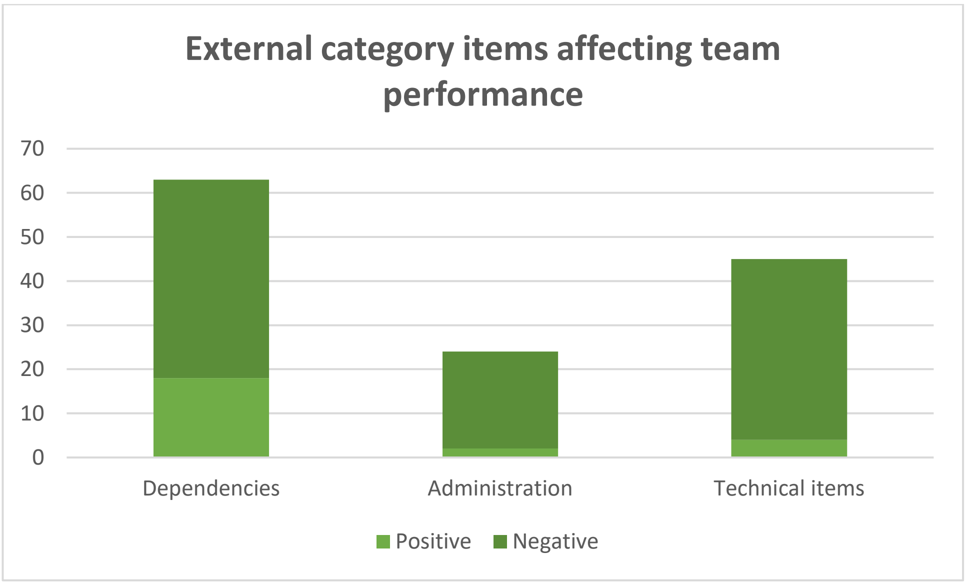


Figure 10. External items affecting team performance.

Only 18% of the items identified to the *external* main category were considered to have a positive impact on the teams' performance, while 82% had a negative impact. Environment where all SAFe trains in scope of this study are operating is complex. There are lot of dependencies outside the teams and the trains. There is a lack of independent test environments. Test data quality or test environment itself, for which the single team or train cannot affect to is one big obstacle to improving team performance. Considering these it was expected that this category would be dominant with negative items.

Dependencies subcategory had the largest number of items found in the *external* main category with clearly the most positive items as well. 75% of the positive external items were in this category. When large software projects involve large amount of dependencies, especially from different organizations which are participating in the delivery, it is not surprising that those dependencies are often mentioned in the sprint retrospectives. They can either greatly hinder or help the team in reaching their objectives. Negative effects can have the greater effect which explains why 71% of the items in the dependencies subcategory are considered negative. Slow and fast responses from the teams which Agile team is dependent on, were the most common items categorized to Dependencies subcategory.

After dependencies the next largest subcategory was found to be Technical items with 45 total items of which 9% where positive and 91% were negative. These items are completely out of the teams' control and not having working test environment for example can completely block the team from progressing with their tasks. Considering the number of sprints observed for this research (85) and the amount of negative comments from the retrospectives data related to technical challenges, it has a substantial negative effect on team performance. The most common item assigned to Technical items subcategory were items related to the development environment, which were almost always negative. Improving the development tools and environments would greatly reduce the negative impact they are having on the teams' ability to deliver working products to the client.

Administration subcategory had only a total of 24 identified items of which 8% were positive and 92% negative. As the category itself can be found to be quite negative in tone it is not surprising that the number of negative items is so high compared to positives. Some of the negative items assigned to this category were "access rights issues" and "software release management unclarities". Bureaucracy and mandatory processes can be considered negative in nature if the teams are not trained enough in them, especially common processes. If either of the parties is not fully committed to following agreed processes they will most likely be considered to hinder the performance of the teams, as they do not get the full benefits of everyone committing to same rules.

5 DISCUSSION AND CONCLUSIONS

The goal of this research was to identify factors which affect the performance of the Agile teams' and the conclusions presented in this chapter are the factors identified. Categories with the most number of items were considered to have the most impact on the performance of the teams.

According to collected Sprint retrospective observations, it seems that positive impact of Agile team performance come from the actions to which the team can affect internally. Most negative effects to Agile team performance are related to external issues. Team performance is negatively affected when they are acting in an environment where there are lots of dependencies, technical issues, lack of processes and complex multi-organization train and team structure.

According to the study most of the sprint retrospective observations were related to the observations which were external of the team. Result was surprising, because before the research, it was expected that most of the sprint retrospective observations would be related to the items which are categorized in the internal main category.

49% of identified team performance impact factors in this research had a positive impact on the team's performance. 87% of items which were considered as having a positive impact to team's performance were related to the team's internal factors which they can influence. Biggest positive impacts on the teams' performance came from team coordination, goal orientation and team learning. 13% of positive team performance items were related to the external environment where the teams are operating. Mostly positive impact from external factors to team's performance came from well managed dependencies with other organizations. Mainly external factors had negative influence on team's performance.

51% of collected Retrospective comments were identified as negatively affecting the performance of those teams. 58% of those were related to external dependencies management, administrative processes or technical issues. 42% of the negative observations were categorized in team's internal factors. Those were related to team's internal factors which also have the biggest positive impact to team's performance: team coordination, goal orientation and team learning.

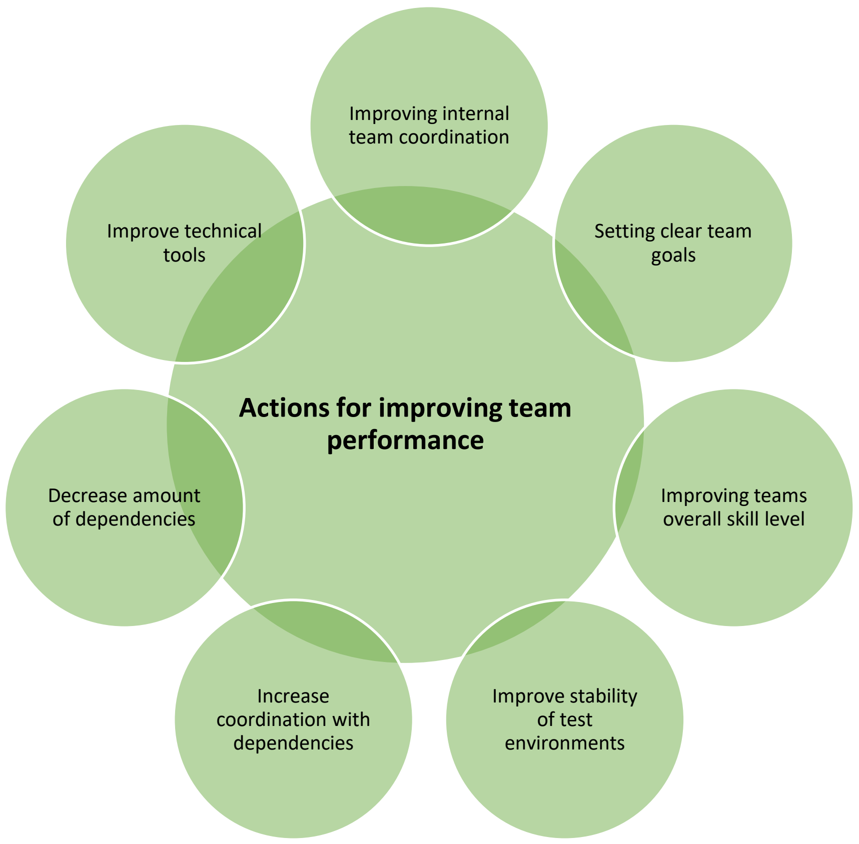

Improving internal team coordination
Improve technical tools
Setting clear team goals
Actions for improving team performance
Decrease amount of dependencies
Improving teams overall skill level
Increase coordination with dependencies
Improve stability of test environments

Figure 11. Actions for improving team performance

Teams can improve their performance with internal actions. Improvement in coordination within the team, having clear goals for the team and continuously improving team processes/skills will have the biggest impact in increasing team performance. Concrete actions that can be taken are to improve the estimation of stories and dedicating IP-sprint (last sprint of PI, planning sprint) for planning which will make the sprint plans more accurate. Internal knowledge transfer within the team can help to improve the skills within the team. Often some persons are more fluent in other techniques or skills and others know something else. By sharing the knowledge within the team can greatly increase the overall skill level in the team.

External factors which have negative impact to team's performance demand improvement actions from all parties in the environment where the team is acting. Dependencies and technical items had the most negative impact on the agile teams’ performance. They can completely prevent the team from being able to proceed, for example if a test environment is not usable or a team which is providing something that the Agile team needs is late. Having stable technical environments and tools for the development and better coordination or less dependencies could significantly reduce the negative impacts on the teams’ performance.

It should be considered that the data used for the research was made by the Agile teams’ themselves as part of the sprint retrospective ceremonies. In SAFe it is used to gather things that the team itself can improve in, but as the development environment in these Accenture teams doesn’t fully follow the SAFe model a lot of outside impacts are also considered by the teams. In full SAFe model all the different levels are controlled by the same organization which can reduce the negative effects of dependencies on the teams. That might partly explain the large number of negative items in the external category as outside impact usually doesn’t assist the teams’ in reaching their goals. The research scope could be increased to include also other teams’ working as within the SAFe model for the same client. That could give even more broader view to the impacts on the teams’ performance. As the full SAFe model is not used in this development environment it could be interesting to see the results from a similar research done for agile teams’ part of an enterprise using the full SAFe framework.

6 SUMMARY

The goal of this research was to identify factors which affect Agile teams' performance either positively or negatively. The research was conducted within one Accenture client and all SAFe teams with only Accenture employed members were part of the research. Research data was gathered from sprint retrospective materials created by teams after each sprint. Data was gathered from a total of 85 sprints, from 11/2015 to 4/2018. Items considered to have an impact on the teams' performance were picked up from the data and then assigned to categories based on the literature assembled.

A total of 366 items affecting Agile teams' performance were identified in the research. Of those 366 items, 49% were categorized as having a positive impact and 51% as having a negative impact. In addition to splitting the items into positive and negative, they were also assigned to categories based on where the impact to the team comes from, internal or external. 64% of the 366 items were categorized as having impact from within the team, to internal category. The rest were categorized to external category. Majority of the positive items found in the research were found to be in the internal category and the external items were mostly negative.

Internal factors, such as team coordination, goal orientation and team learning had the biggest positive impact and increasing those factors would increase performance the most. Biggest negative impact on the teams came from outside the teams, from external factors. Dependencies and technical items were found to be the biggest hindrance on the teams' performance. Improving coordination with dependencies or reducing them if possible and improving the stability of the technical items, such as development environments, could greatly increase the performance of the agile teams.

REFERENCES

Accenture, (2018). *About Accenture*. [online] Available at: https://www.accenture.com/fi-en/company [Accessed 8 August 2018].

Agile Alliance, (2001). *Manifesto for Agile Software Development*. [online] Available at: https://www.agilealliance.org/agile101/the-agile-manifesto/ [Accessed 3 March 2017].

Agile Alliance, (2001). *12 Principles Behind the Agile Manifesto.* [online] Available at: https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/ [Accessed 3 March 2017].

Anderson, D. and Carmichael, A. (2016). *Essential Kanban Condensed*. Seattle, Washington: Lean Kanban University Pres, 89.

Coad, P., Lefebvre, E., De, J., (1999). Java Modeling In Color With UML: Enterprise Components and Process. New Jersey: Prentice Hall PRT, 218.

Choudhary, B., Shanu, K. R., An Approach using Agile Method for Software Development, In: *1st International Conference on Innovation and Challenged in Cyber Security (ICICCS),* February 3-5 2016.

Cooke, J L. (2012). *Everything you want to know about Agile: How to Get Agile Results in a Less-Than-Agile Organization.* 1st ed. [ebook] IT Governance Publishing, 208. Available at: http://www.jstor.org/stable/j.ctt5hh467 [Accessed 20 March 2017].

Dingsøyr, T. and Dybå, T. (2012). Team Effectiveness in Software Development: Human and Cooperative Aspects in Team Effectiveness Models and Priorities for Future Studies, In: *5th International Workshop Cooperative and Human Aspects of Software Engineering* CHASE), pp. 27–29.

Dingsøyr, T., Fægri, T.E., Dybå, T., Haugset, B., Lindsjørn, Y. (2016). Team Performance in Software Development: Research Results versus Agile Principles. *IEEE Software*, 33(4), pp. 106-110.

Dingsøyr, T., Lindsjørn, Y. (2013). Team Performance in Agile Development Teams: Findings from 18 Focus Groups. In: 14th International Conference on Agile Software Development. Vienna: Springer-Verlag Berlin Heidelberg, p. 253.

Gebert, D., Boerner, S., Kearney, E. (2006). Cross-Functionality and Innovation in New Product Development Teams: A Dilemmatic Structure and Its Consequences for the Management of Diversity. *European J. Work and Organizational Psychology*, 15(4), pp. 431–458.

Inno.com, 2016. *How does the SAFe 4.0, the Scaled Agile Framework, benefit the Enterprise architect?* [online] Available at: https://inno.com/uncategorized/safe-4-0-scaled-agile-framework-benefit-enterprise-architect/ [Accessed 1 March 2018].

Johnson, J. (2002). Features and Function Usage, In: *XP 2002*, Sardinia.

Jewett, S. (2008). Lean Software Development, In: *Systems & Software Technology Conference*, 2010.

Kang, H.R., Yang, H.D., Rowley, C. (2006). Factors in Team Effectiveness: Cognitive and Demographic Similarities of Software Development Team Members. *Human Relations*, 59(12), pp. 1681-1710.

López-Martínez, J., Juárez-Ramírez, R., Huertas, C., Jiménez, S., Guerra-García, C., Problems In The Adoption Of Agile-Scrum Methodologies: A Systematic Literature Review, In: *4th International Conference In Software Engineering Research And Innovation*, April 27-29 2016.

Malone, T.W., Crowston, K. (1994). The Interdisciplinary Study of Coordination. *ACM Comp. Surveys*, 26(1), pp. 87-119.

Markkula, J., Rodríquez, P., Oivo, M., Turula, K., Survey on Agile and Lean Usage in Finnish Software Industry, In: *ACM-IEEE International Symposium on Empirical Software Engineering and Measurement,* September 20-21 2012.

Moran, A. (2015). Managing Agile. Zurich: Springer, 266.

Mudrack, P.E. (1989). Defining Group Cohesiveness: A Legacy of Confusion. *Small Group Research*, 20(1), pp. 37–49.

Peterson, D. (2015). *What is Kanban?*. [online] Available at: http://kanbanblog.com/explained/ [Accessed 17 October 2017].

Poppendieck, M. and Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit.* New Jersey: Addison-Wesley, p. 240.

Pyritz, B. (2003). Extreme Programming in the Telecommunications Domain. *Bell Labs Technical Journal,* 8(3), pp. 97-100.

Salas, E., Sims, D.E., Burke, S.C. (2005). Is there a "Big five" in teamwork? *Small Group Research*, 36(5), pp. 555-599.

Scaled Agile, (2018). *SAFe 4.0 for Lean Software and Systems Engineering.* [online] Available at: http://v4.scaledagileframework.com/ [Accessed 1 March 2018].

Scaled Agile, (2016). SAFe 4.0 Introduction, In: *A Scaled Agile, Inc. White Paper*, 2016.

Sharma, S., Hasteer, N., A Comprehensive Study on State of Scrum Development, In: *International Conference on Computing, Communication and Automation (ICCCA2016),* April 29-30 2016.

The Burndown, 2018. *SAFe 4: PI Planning Step-By-Step.* [online] Available at: http://theburndown.com/2018/01/12/adaptive-release-planning/ [Accessed 4 March 2018].

Van Baelen, R 2016, *Agile Manifesto Wallpapers*, photograph, viewed February 2017, <https://rafvb.wordpress.com/2012/10/18/agile-manifesto-wallpapers/>.

Womack, J. and Jones, D. (2003). *Lean Thinking: Banish Waste and Create Wealth in Your Corporation, Revised and Updated.* New York: Simon & Schuster, 396