Table of Contents

CHAPTER 1: THE FIELD OF STUDY

1.1: Introduction

It has become a cliché to state that the world is more uncertain than ever before. Several indicators, such as the number of patent applications filed, as well as the rise in registered start-up companies per year during the previous fifty years were in a study found to back up this statement. In the same study, computer software were found to be the third most uncertain of all industries (Dyer, Furr, & Lefrandt, 2014). This implies how the software industry is highly competitive and turbulent. From a development perspective, this means software companies indeed have to deliver more innovative and cost-effective solutions, while simultaneously pushing the boundary for achieving shorter lead-times (Peterson, 2010).

As a response to the same trend of uncertainty, this already led to a rise in Agile light-weight software development practises in the 90's, which were essentially based on customer collaboration as well as short feedback and development cycles. These Agile practices were in sharp contrast to the dominant plan-driven ways of organizing software development, such as the waterfall development model. Whereas this waterfall model represented an attempt to reduce uncertainties by crafting a predictable and well-planned process, the Agile practices embraced the turbulence as a fundamental working condition, in a rather controversial manner (Highsmith, 2002). Much water has flowed under the bridge since, and in current times, the top-tier consultancies' frequent usage of Agile as an arbitrary buzzword in line with disruption, seems to underline how Agile practices have become indeed very popular. This is a fact that has been documented by a recent survey which states a more than 50 % adoption level of Agile principles and practices within the software industry (Strode, 2015). While this gradual adoption of Agile has been taking place, the concept of Lean has since Poppendieck & Poppendieck (2003) published the first book on the subject, started to influence the conduct of software development on a global scale. With its origin in manufacturing and chain-supplies, Lean has demonstrated very convincing results for more than fifty years, in its end-to-end process focus on removing waste, which is defined as everything not adding value to the customer. As much as the separate bodies of literature on Lean and Agile are very extensive, the literature on combining the concepts often take a nascent high-abstractive form. This is a problem, as the combination of the concepts could potentially prove to increase the chances of survival in an uncertain industry. With the increasing number of software organizations and professionals utilizing both concepts, this calls for a better understanding of the similarities and differences on a more low-practical level, and further how and under which circumstances, the concepts in turn can complement each other.

1.2: Objective

This study aims to contribute and offer recommendations to practitioners and the current state of the literature on how to combine the concepts of Lean and Agile within software development. It will thereby seek to uncover the sum of its principles, practices, tools, activities and performance metrics, with the aim of deriving knowledge from a combined academic and practitioner-based body of sources. Below is stated the research question, which will steer the progress of the study:

How are the concepts of Lean and Agile combined by academics and professionals within software development processes?

In addition, the four following sub-questions all related to software development processes, have been constructed in order to further specify the scope of the study:

A. How are the foundation, principles & practices of Lean and Agile related?
B. What empirical findings and suggestions exist for further research on combining Lean and Agile?
C. How are software professionals in turn combining Lean and Agile practices, tools, activities and performance metrics?
D. How and why are software teams applying Lean and Agile in different situational contexts?

The sub-questions A and B are related to the current academic literature on Lean and Agile in relation to software development. In contrast, based on this knowledge, sub-question C and D approach the subject from a practitioner-based point of view – sub-question C emphasizes the broad consensus in the ecosystem, while sub-question D seeks to understand the subject in its situational context.

1.3: Delimitations

As already stated, the scope of the thesis is to encompass elements with direct relevance for software based new product development – in other words, the study of the craft of software driven applications/services. Nevertheless, the foundation of the concepts of Lean and Agile, will nevertheless be included in the literature review in part 1, due to the acknowledgment of the importance of understanding the evolvement of the concepts. Essentially, the study will

not go in depth with elements of combining Lean and Agile outside the realm of the organization, including external aspects such as its widespread diffusion within the field of chain-supply management. The integration of external parties such as users for product feedback or co-development will nevertheless be analysed in its relation to Agile peer-review as well as the Lean start-up methodology's application within software development. Although production will be included as an element of describing the evolvement of Lean's application to product development as well as its relation to DevOps, it will only be in a stringent relation to delivering the best conditions for a successful product development process. In the same spirit, the widespread Kanban, will be thoroughly integrated, but not encompass its properties in relation to evolutionary change management according to the methodology of David J. Anderson (2010). Although security and compliance procedures will to some extent be mentioned in relation to how it might set the fundamental boundaries for the development process rhythm, its greater implications for product development processes will not be analysed in depth. The study will also not include any cognitive motivational aspects on an individual subject level of analysis. Finally in regards to the various Lean and Agile practice/schools, the study will only go-in depth with a limited number of the ones found to be the most commonly used.

1.4: Organization of the study

The thesis is divided into 3 main parts: a literature review, a LinkedIn content analysis and a multiple-case study. The aggregated findings from the three main chapters will lastly be discussed in relation to the academic and managerial implications according to the study objective. With the starting point in the literature review, the following two chapters are in that way designed with the purpose of progressing and qualifying the findings of the previous.

Part 1: The initial literature review establishes clear boundaries and provides insights in regard to the evolvement and status quo of the academic knowledge of the foundation, principles and practices within the field. Current academically neglected areas are also presented following the guidelines stated in recent and unanswered recommendations for future studies.

Part 2: Based on these appeals, the LinkedIn content analysis can be summarized with its compliance in following the assumption that the field is a nascent research area (Al-Baik & Miller, 2014; Wang, Conboy, & Cawley, 2012). With access to more than 100.000 IT professionals, it illustrates how practitioners in turn combine practices, tools, activities and

metrics in their respective software development processes. In that sense, the content analysis is more than understanding the practical software process, but it is also illustrative of which of the process aspects that are clearly under-represented in academia (Dybå & Dingsøyr, 2008; Sjøberg, Dybå, & Jørgensen, 2007).

Part 3: Although the LinkedIn review delivers rich data to understand the dissemination and practical usage of what it presents as a *best practice* of software development, it generates vast insights on how to combine the concepts in specific contexts. For this reason, a multiple-case study was conducted in order to analyse how the concepts Lean and Agile principles and practices are utilized in four distinct software development companies within a selection of their projects. Furthermore, the interviews conducted emphasize on understanding how tools, activities and metrics are applied and accordingly why the software development process in question has been selected.

CHAPTER 2: Research methodology

2.0: Overview

This chapter begins with a brief categorization of the current state of the field, followed by an elaboration of the levels of abstractions within the study. An introductory graphical timeline of the evolvement of Lean and Agile is then presented, followed by an examination of the methodological process of the literature review, Linkedin content analysis and the multiple-case study.

2.1: Defining the current state of theory

Following Edmondson and McManus (2007), the research design in any given field as a rule of thumb needs to fit the current state of theory and research. To them, research is an evolutionary process in which academic fields (due to solid academic work) over time floats through the states of being nascent, intermediate, and mature. In the nascent state, the research question should be open-ended and initiate the collection of qualitative data which should be interpreted for meaning, in order to formulate suggestive theory (Edmondson & McManus, 2007). Based on these suggestive patterns, the classic intermediate state research question proposes relationships between new and established constructs. The data collected can be a mixture of qualitative and quantitative ones, and the derived provisional theoretical contribution integrates previously separate bodies of knowledge (Edmondson & McManus, 2007). Finally, if the academic interest and collaboration continue, the mature state of a field is having research question which are focused, perhaps through the identification of specified hypothesis related to the existing constructs. Focused questions thus require focused measures, which reflect that the scope of the empirical collection is primarily oriented towards quantitative data. The testing and/or validation of existing patterns thereby contribute to the existing body of knowledge and add new mechanisms or establish boundaries to already established theories (Edmondson & McManus, 2007).

As can be seen in the following references, Lean and Agile have since 2008, with the latest references in 2014, both been classified to be nascent within the discipline of software development (Al-Baik & Miller, 2014; Dybå & Dingsøyr, 2008; Wang et al., 2012). Several researches in the discipline have followed Dybå & Dingsøyr's (2008) suggestion of studying the phenomena through an explorative qualitative approach, also by the authors iconic pronounced as a *"backlog of research problems to be solved"*(p. 852). Although it has not been possible to find any authors stating Lean and Agile to be within an intermediary state,

patterns in the literature review illustrates how research has begun to utilize hybrid data in order to increase the quality and validity of previous suggested theory.

To fully understand this current state of the field, it is required to have a sufficient level of contingency to available prior studies conducted (Edmondson & McManus, 2007). For the same reason, the findings and proposals for future studies that will be uncovered in the literature review will be assessed in order to craft a progress in the study that will contribute to the current state of the field. The term *theory* may be defined and evaluated in many ways. Software development lies in the applied software engineering discipline, and should have practical value and congruence within the software industry (Dybå, Sjøberg, & Cruzes, 2012). Theory must in that sense be generated and, from my perspective, also used with the purpose of proving fit for real life decision-making, in which every setting demands a local adoption to be useful. This acknowledgment and success criteria have guided the organization and logical progress of the study.

2.2: Levels of abstraction within the study

Inspired by (Modig & Ahlstrom, 2012), the thesis conducts its analysis within the following levels of abstraction, as depicted in figure 1.

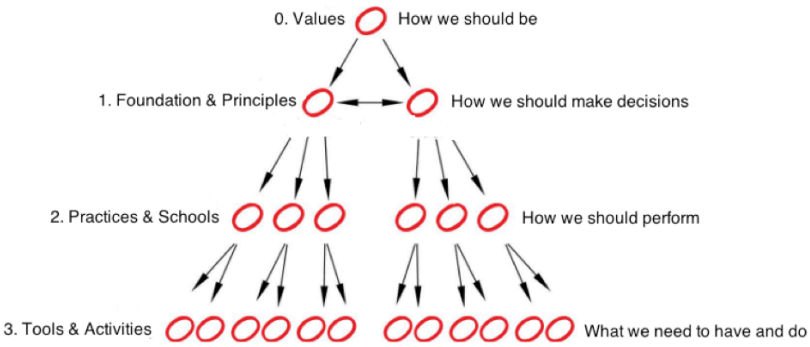


Figure 1: The four levels of abstraction. Inspired by (Modig & Ahlstrom, 2012).

Furthermore, performance metrics will be added as an independent body within the third level of abstraction. This is based on the rationale that performance metrics nowadays can be managed automatically within software tools, such as virtual Kanban boards, but at the same time also can be considered as the conduct of an activity in it self. For future development of the framework of levels of abstraction, fellow academics might consider to add performance metrics as an independent fourth level of abstraction encompassing: "*How have we performed*

and how can this be improved". For future development of the framework, it might also be an advantage to split tools and activities into further subcategories.

2.2.1: Values

It is important to note that values are stated as the baseline level of 0, which will not be an active unit within the analysis. Hence the concepts, in line with the findings of Peterson (2010), are assumed to have shared goals. As will be examined in the literature review, this is from my perspective due to the historical integration Lean and Agile has had (Al-Baik & Miller, 2014; Highsmith, 2002; Poppendieck & Poppendieck, 2010). The literature review led to the identification of the following timeline (figure 2), which offers an introductory overview of this shared evolution Lean and Agile have encountered.

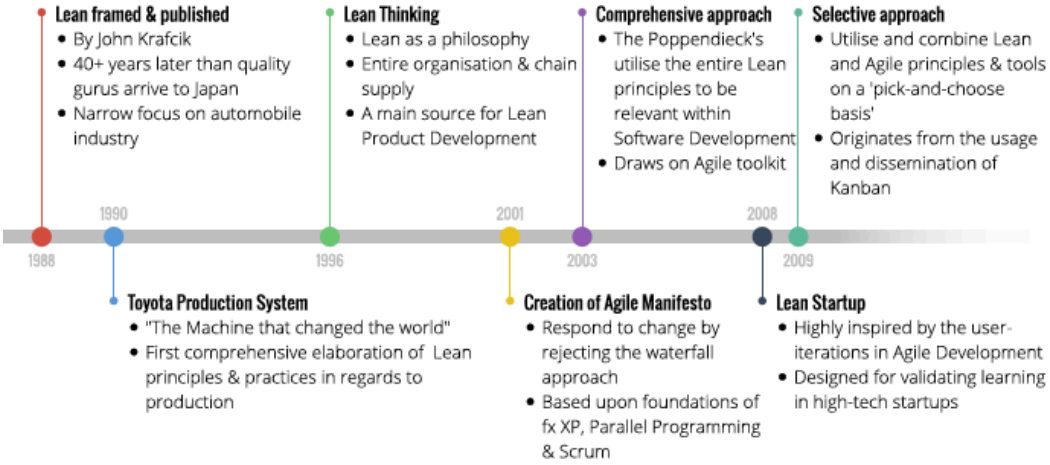


Figure 2: Timeline of the evolvement of the concepts. Own creation

2.2.2: Foundation & Principles

The emphasis on an analysis of the foundation investigates the historical context, which in the study is essential to understand the origin and evolution of the concepts on all the levels of abstraction. The principles are in essence for both Lean and Agile, very explicit stated and will be examined in the literature review. In the original framework of (Modig & Ahlstrom, 2012), which has its origin on analysing Lean manufacturing, Jidoka & Just-in-Time are noteworthy principles, as they have been defined by Tachi Ohno to be two main pillars of distinction. Jidoka means *autonomation* of detecting errors, which also means automation with a human touch. Just-in-Time describes Toyotas way of organizing a pull production system, in which production is not initiated before a demand triggers the process. Even though it can be agued these principles are also main pillars in Agile development, I have decided not to include the principles as main principles in order not to de-order any principles over orders.

2.2.3: Practices and Schools

While principles are guiding ideas and insights about a discipline, practices are the methodological output that narrows down the path to carry out the principles (Poppendieck & Poppendieck, 2003). Principles can thus be viewed upon as universal, however having different forms applied to environments in practice. This practice to transform principles is context dependent, and such thing as a *best practice* is therefore only relevant for a specific context (Poppendieck & Poppendieck, 2003). According to the Poppendiecks, this is namely true within software development, which is a truly broad domain encapsulating the spectrum from "*building a website to sending a satellite into orbit" (p. xxiv)*. Lean and Agile are indeed both very comprehensive and debated concepts. In time they have been categorized into different practices and grouped into schools. During the study, it will become evident, how practitioners in turn choose on a problem-solving pick-and-choose basis.

2.2.4: Tools, Activities & Performance Metrics

Defined by its scoop of relevance by the previous levels, the tools and activities are more visible and to a larger degree possible to assess by conducting performance evaluations. Due to the tangible nature of tools & activities, they are more distinct to identify, as they constitute the practical level of abstraction. Nevertheless, the dynamics can be blurred out, as some tools such as Kanban and Scrum become sufficient complex and widespread, so they begin to gain status among academics and practitioners as practices. Although the tools & activities will not be fully uncovered before the LinkedIn content analysis, the literature review will include the findings from previous studies on using performance metrics within Lean and Agile software development.

2.3: Literature review methodology

The literature review is inspired by the prescriptions of conducting reviews within the field of Software Engineering & Development according to (Brhel, Meth, Maedche, & Werder, 2015; Dybå & Dingsøyr, 2008; Kitchenham et al., 2010; Petersen, Vakkalanka, & Kuzniarz, 2015). As the review does not include statistics of annual releases or encompass a systematic coding of selected articles, it can be argued to be limited to being a semi-systematic review. The purpose of the review is to present the foundation and evolvement of the field, and to investigate the principles and practices, which will be further examined within the LinkedIn content analysis. The review aims to answer the sub-questions of A) how are the foundation,

principles & practices of Lean and Agile related and B) what empirical findings and suggestions exist for further research on combining Lean and Agile.

2.3.1: Search, selection process and coding

The search progress took place on 5th of May 2016 through CBS' Library's extensive *Libsearch*, in which the selection was processed according to figure 3:

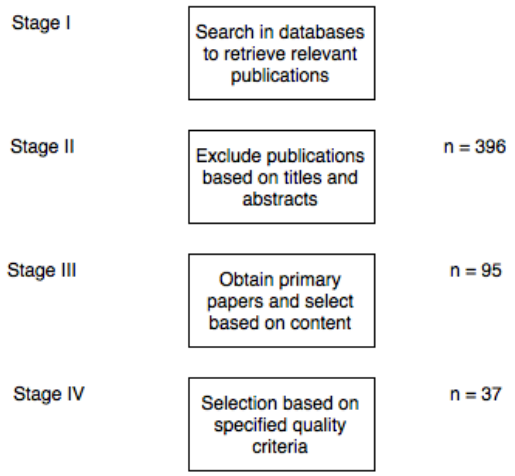


Figure 3: Literature review methodology. Own creation

As there were more than 20.000 matches relevant to the start criteria's of the search phrases, only literature written in English and having the search phrase stated in title or its associated tags were selected. Furthermore, only literature relevant for Lean & Agile software development were selected, with the exception of literature presenting the foundation and evolvement of Lean. The literature selected was also assessed on its originality; that is, whether it has been following proper academic methodological standards and appropriately academic peer reviewed. The search process was initiated according to the phrases in figure 17, located in appendix 7.2.2.

The final selected literature was then grouped into the categories of Lean, Agile or Combined. In the presentation the topic and field of study of the selected literature were noted. With the intention of demonstrating several different perspectives on the foundation, principles and practices in a qualitative manner, no further systematic coding of the literature was conducted. All abstracts, conclusions and recommendations for future studies were in the initial categorization of the literature read through twice. Hereafter the selected literature was, with the exceptions of the selection of chapters within the selected books, all read thoroughly at least one time, while notes were taken in another document.

2.4: LinkedIn review methodology

LinkedIn was chosen for the content analysis, due to the platform's high level of engagement of experienced IT professionals. LinkedIn has since May 2015 restricted their API's, in effect blocking access to 3rd party analytics software such as Nvido (Ncapture). In essence, this means that contrary to conducting data analytics on platforms such as Facebook and Twitter, LinkedIn data have to be retrieved manually through scrolling down and updating browser data and then conducting searches in the browser by Ctrl + F.
As LinkedIn is a networking platform with content being primarily business-oriented and members having often very detailed profile information, it is an interesting site for conducting research. It has to be noted all open social media platforms are biased, as they are according to figure 4, somewhere within the *public sphere*.

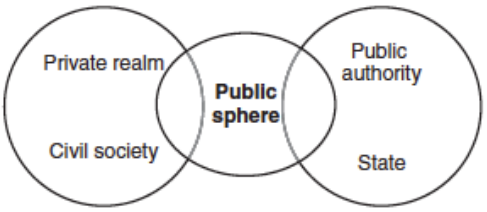


Figure 4: Social media platforms as a Public sphere (Cantijoch, 2014).

It is therefore relevant to mention how the 1) professional stake and incentive and 2) screening of members in professional groups, might be able to provide an incentive for the members to express and share content that are career enhancing for their professional lives. A main presumption in the LinkedIn study is the findings on LinkedIn might, to a stronger degree, portray the benefits rather than the negative results experienced in combining Lean and Agile. This was not accurate, as a lot of the group members in turn had little or no barrier against sharing and openly analysing their own failed development projects.

On a search on LinkedIn on 25th May 2016 on "Lean and Agile", there was a result of 175 Groups in English. Before the search I had previously applied for all relevant groups, in order to gain an overview of how many of the relevant groups I could manage to become a member of. The process proceeded accordingly to figure 5:

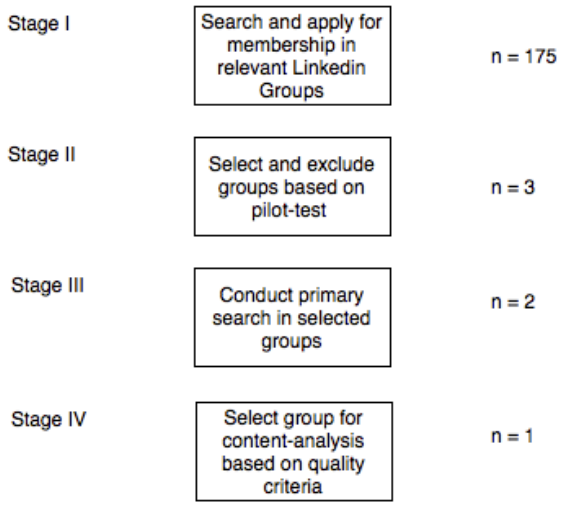


Figure 5: LinkedIn review methodology. Own creation

The three initial groups were selected through the following criteria:

1. Encompassing both Lean and Agile Software development
2. Having a high number of active practitioner members, in which the content and engagement activity on the groups are decentralized.
3. Moderators of the groups allowing my personal LinkedIn profile access to the group.

The three initial selected groups were

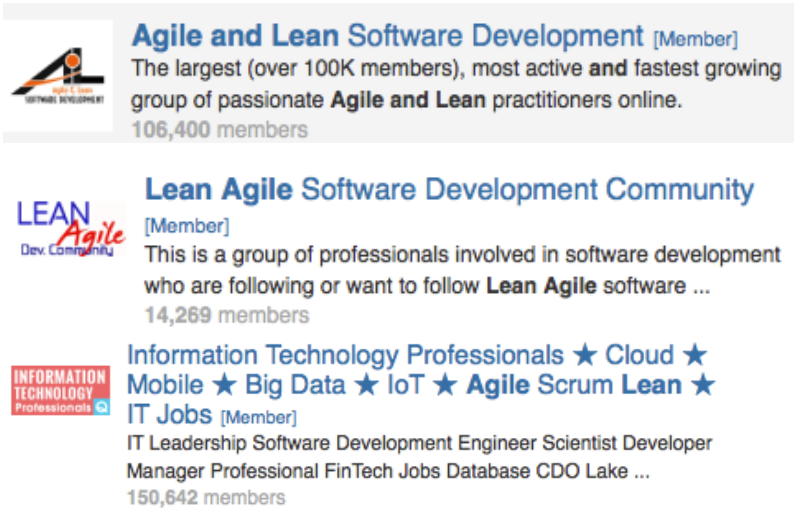


Figure 6: Initial selection of groups. Screenshots from LinkedIn.

This resulted in a combined number of 271.311 members, of which all have been approved and somewhat screened prior to joining the groups by the respective community moderators. It should be noted it has unfortunately not been possible to grant access to any metadata describing the average contribution level of each member, which would in turn go further into the activity level for each of the members.

After an initial pilot-searching on Lean and Agile + search phrases, the 'Information Technology Professionals' group were excluded from the analysis, due to very low content associated with Lean and Agile. This is despite this group were by far the most active, although with a high degree of content related to centralized marketing or other commercial purposes. The total number of groups members were therefore 120.669 at the time of the content analysis.

2.4.1: Search, selection process and coding

As a progression of the output of the literature review, the purpose of the LinkedIn review was to uncover the dissemination of practices, as well as to identify and understand the relevant tools, activities and metrics software professionals in the group were utilizing. This goal was the guiding foundation in defining the search phrases. The search phrases accounted for all identified practices/schools. The timeframe of the search were 4 weeks of post history, crawling the tags, titles and content of the initial post.

In group 1: Agile and Lean Software Development, it counted 96 posts, the majority with related community engagement through comments, created by more than 80 different contributors. In group 2: Lean Agile Software Development Community, this counted 58 posts, of which the majority were related to community engagement through comments, created by more than 50 different contributors. The findings in the groups can be viewed in figure 18, located in appendix 7.2.2.

Besides the strong findings within Scrum and DevOps within group 2, group 1 was selected due to its stronger diffusion of the concepts as well as the fact that it had more than 7 times as many members. It also has to be noted the metrics of Customer Satisfaction, Defect Count, Effort Estimate & Work-In Progress were not included, due to a lack of direct findings in either of the groups. This also goes for Risk Entrepreneurship, Comphrehensive School, Selective School, Customer discovery and Leagile. Feature-Driven Development is depicted as features, as well as requirements due to the nature these two phrases were found to be used similar within the groups. The 0-0 findings of XP indicates how the 1-month sample in the

groups has simply been to small, as this finding is very much in contrary to surveys documenting the dissemination of XP.

According to figure 7, the data was coded hand-driven in a mixture of manual and crowd sourced dictionaries based on the search findings.

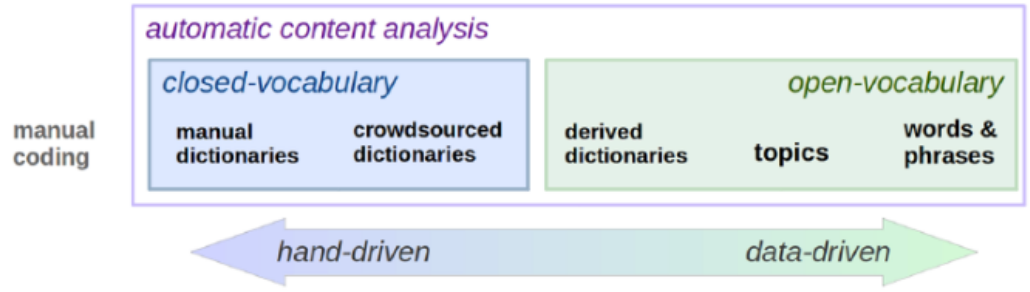


Figure 7: An illustration of open-closed based coding (Schwartz & Ungar, 2015).

This led to the initial selection of 25 posts, of which 5 were excluded due to a low degree of community feedback and/or low profile verification of the creator's profile. The final 20 selected posted can be viewed (including selected comments) in appendix 7.3.

The 20 selected posts have an average of 31,98 comments each, ranging from 9 to 95. Finally it has to be noted the categorization was also considered in regard to representing fair value to the quantitative search findings.

2.5: Multiple case-study methodology

Although accessibility through the snowball method was to a laree degree used for the identification of cases, the final selection of cases were sampled according to presenting maximum variation (Miles, Huberman, et al: 1994). Furthermore the addition of multi-sampling provides a stronger confidence to the findings by including a range of similar and contrasting cases in terms of product/service, organizational size and maturity as well as the influence on the process to utilizing virtual software tools in contrary to being co-located. Within these divergent sampling factors, the how, where and why come in focus to understand the patterns and complexities encompassed within the selection of cases (Miles, Huberman, et al: 1994). Even though generalization is commonly argued to be very inappropiate to strive for within qualitative studies by Guba & Lincon (1981), the leverage of cross-case analytics can nevertheless be argued to enhance the analytical generalizabiliy of the findings (Miles, Huberman, et al: 1994). Thus following Glaser and Strauus (1967 and

1970), the usage of multiple comparison groups can pin down the specific conditions, supporting or challening the likelihood of the finding in hand might occur. To the extent that it can also add more aspects into how the case conditions may be related (Miles, Huberman, et al: 1994). Even though these arguments above may be true for very large samples, it is to my understanding that the four cases sampled in this study should only in their limited size provide inspiration for such further quantitative or analytical variable-oriented analysis.

2.5.1: Case-study design and process

The multiple-case study proceeded accordingly to (Yin, 2009), following the methodological steps of:

1. Designing the case study
2. Preparing the data collection
3. Data collection execution
4. Analysing of the collected data
5. Reporting the findings

First and foremost, the multiple-case study was designed to draw upon the previous findings in Chapter 3 and 4. According to the findings in the literature review, the research design was constructed to highlight different degrees of mature agile teams. Furthermore, it was aimed at investigating the principles, practices, tools, activities and performance metrics that were found to be the most widespread among practitioners in the LinkedIn review, according to the recommendations proposed by (Dybå & Dingsøyr, 2008; Sjøberg et al., 2007).
The preparation and execution of the data collection was based on semi-structured interviews with the selected key informants, in accordance with the recommendations stated by (Kvale & Brinkmann, 2008). In accordance a semi-structured interview guide (see appendix 7.4.1) was crafted prior to the interview, which were aimed at seeking to gain insights to the omnibus and discrete context illustrated in figure 8.

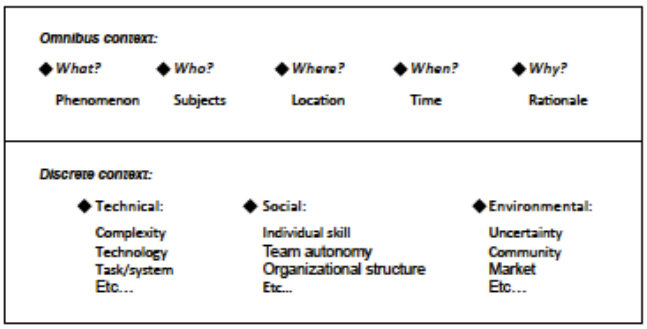


Figure 8: Omnibus and discrete context (Dybå et al., 2012).

As illustrated, the Omnibus context and Disrete context combined embeds technical, social and environmental elements in understanding the characteristics and rationale of the context. The nature of the actual interview-process was in essence unfolding in a conversation format, in which the interview guide was a check-list to make sure the pre-formulated perspectives were included. The interview conversation was in that sense a mixture of improvisational follow-up questions, as well as sticking to embedding the pre-defined questions. The interviews were recorded and transcribed into Danish (See appendix 7.4.2). When all interviews had been conducted and transcribed, they were all read through several types, and then elements from all interviews were selected and translated to English based on its relevance to the scope of the study. Moreover, with the purpose of cross-comparison, the nature of the principles, practices, tools, activities, performance metrics and co-location were identified and grouped into individual charts for each case. All this raw data describing the full development phases within the four cases, were then on an individual case-by-case basis grouped into the three categories of task definition and estimation, the development process and the testing and evaluation. The purpose of this initial coding step was to further gain insights into the similarities and differences among the cross-case findings of the selected raw data. These three categories were eventually for communication purposes, changed into the four subgroups of 1) finding & validating ideas, 2) day-to-day development, 3) testing, integration and deployment & 4) performance evaluation and process improvements.

The rationale for the following coding procedures was to present the findings to the reader in the most accessible and informative way, while delivering fair value to the general patterns in the sum of the raw data collected. Beyond the specific descriptions and findings, the presentation of findings thus sticks to portraying the development process in its full length, drawing from good and bad experiences identified within the four different cases. Moreover, the rationale or explanation of the current process layout is embedded when directly stated by the interviewees.

CHAPTER 3: LITERATURE REVIEW

3.0: Overview

This chapter will begin with a brief examination of the paradigmatic evolvements from craftsmanship to mass-production, which from my perspective is essential to begin understanding Lean, as well as the rationale behind the creation of the Agile and software craftsmanship manifesto. Next, the literature review will begin with a chronological presentation and analysis of the literature in relation to Lean, Agile and the combination of the concepts. Throughout the review, **bolded phrases** followed by (**number** and **letter**) will indicate if it is a principle/practice, which has been included in the final chart, in order to synthesis the findings in a structured manner. Figure 19 presents in appendix 7.2.2. an overview of the literature selected for the review.

3.1: From craftsmanship to mass-production

Before the industrial take-over, the production and development activities were somewhat blurred out, and were typically associated with artisan's discipline of craftsmanship[1], characterized by a larger degree of individual elements of creativity and customization. With the rise of new methods of standardization, the craftsmanship-manufacturing of cars would not be a matter of producing and assembling modular parts, but instead producing one item at a time by making the relevant components (Womack, Jones, & Roos, 1990). In the 1920's, in the beginning of the era of cars, Henry Ford had great success in transforming this approach into a concept of mass-production of cars through assembly lines. Henry Ford had studied the work of the *father of industrial efficiency* Frederick W. Taylor, and based Ford's production system on the main pillars from his scientific literature (Nicoletti, 2015). One of these pillars of the assembly-line production was the acknowledgment of Taylor's scientific objectivism, in which the afore-mentioned individual divergence and unreliable attributes of craftsmanship needed to be minimized as much as possible. As figure 9 illustrates, the efficiency advantages of this highly standardized mass-production system was indeed very convincing.

[1]Craftsmanship can be defined as the very elite of the human attributes of knowledge and skills required for performing a task of transferring material into a product (Wikipedia).

Mins. of effort to assemble	Craft production (Fall 1913)	Mass production (Spring 1914)	Percentage reduction in effort
Engine	594	226	62%
Magneto	20	5	75%
Axle	150	26.5	83%
Major Components into a Complete Vehicle	750	93	88%

Figure 9: Craft- and mass production compared (Womack et al., 1990).

Without going in depth with these scaling advantages of reducing the marginal cost of production per item, still it should be noted how this was clearly achieved by the ingredients of component standardization and the specialized division of labour. In essence the majority of the workforce needed would be unskilled and cheap labour, which were only given a very specific defined work task such as performing the same screw action repeatedly. The same logic was applied on the machines, which were very inflexible in terms of adjusting into producing future improved parts (Womack et al., 1990). To perform quality control, a few specialists were hired to oversee the occurrence of defects by looking at the units at the end of the assembly line, which resulted in defects being acknowledged at the very last steps (Womack et al., 1990). As the whole production system was geared towards reaching the highest incremental productivity of units produced, the incentive structure did not facilitate a great emphasis on line-employees' thoughts on quality control and process improvements. As much as this prioritization of reducing the (majority) of the human cognition to its absolute minimum was the strength of the whole system, it also proved to be its Achilles' heel in the long term.

3.1.1: Towards company-wide quality control

A few years later in the post-war years of the 1940's, a few counter-scientific points of viewing quality attempted to influence the American production environment. The Ford doctrine of mass-production was more competitive than ever in the Western world, and it was impossible to convince anyone about the space and need for improvements (Nicoletti 2012: 36). This was far from the case in post-war Japan, which in their total destruction and industrial reconstruction sought new paths. These scientists, who in Japan were named the three American quality gurus[2], succeeded in translating their innovative ideas into what were to be the Japanese way of doing business. A fundamental game-changing aspect in the guru's

[2] Who were W. Edwards Deming, Dr Joseph M. Juran & Armand V. Feigenbaum

literature was the guiding principles concerning the importance of quality management as a competitive factor. This needed to be strategically applied by the senior management, in order to promote actions of improvement. Finally, they suggested statistical methods as being crucial, but only as a mean to complement the qualitative evolution of the organization (Nicoletti 2012). In turn, these principles gave among other accomplishments birth to the P-D-C-A model[3], which dictated how organizations need to address a continuous improvement approach, a logic, which has since been framed *Kaizen.* This cycle of improvement was based on an aim to continuously improve quality; that is quality defined from the attributes, which was visible to the customer. Profit was not an end goal, but a natural consequence of delivering a sufficiently high amount of quality. Quality was a concern for all employees in the organization; it should therefore, from a strategic point of view, be embedded in the culture of the entity (Nicoletti 2012: 37).

3.1.2: From Lean production to Lean Thinking

Most people familiar with Lean, associate it with the accomplishments of the famous *Toyota Production System* (TPS) and vice versa (Karlsson & Åhlström, 1996; Modig & Ahlström, 2012). The western interest in the field originated in 1979, when the Japanese car manufacturers began acquiring significant market-shares previously held by The Big Three, being Ford, GM and Chrysler. As a result, the *International Motor Vehicle Program* (IMVP) was launched as a research consortium on MIT, in order to understand the new industry dynamics (Womack et al., 1990).The exact phrase *Lean*, was however not framed before John Krafcik – later to be the CEO of Hyundai - used it in his master thesis in 1988 (Stone 2012). The findings in his thesis were published in the article *Triumph of the Lean Production System*, in *Sloan Management Review*, in which he, through 3 years of first-hand study of automotive production facilities, had discovered a robust and fragile way of running production systems. Due to the negative implications of the word 'fragile', he chose *Lean* to represent the finding of the new efficient way of organizing an automotive production system (Modig & Ahlström, 2012).

Ironically, Krafcik was during the process of his master-thesis supervised by James P. Womack, who 2 years later was a main contributor to the best-selling book, *The Machine that Changed the World (Stone 2012).* This particular book had long been under way since the establishment of the IMVP research program, and was a five-year in-depth study on the

[3] **Plan:** planning goals and conditions, **Do:** perform the activities, **Check:** measure improvement in process and activities, **Act:** work to improve

processes deployed at the entire Toyota organization, as much as an analysis of the evolution of production systems.

3.2: The core principles in Toyota Production System

The Machine that Changed the World, uncovers how Toyota's executive director Taiichi Ohno, known as the father of TPS, managed to apply some of the principles in collaboration with the quality gurus into a comprehensive and operationalized framework. Figure 10 illustrates why it was no wonder Toyota's impressive results in the early 1990's led to an enormous interest among researchers and practitioners.

1989	GM Framingham	Toyota Takaoka
Assembly hours per car	31	16
Assembly Defects per 100 Cars	135	45
Assembly Space per Car	8.1	4.8
Inventories of Parts (average)	2 weeks	2 hours
	Classic mass production	Classic lean production

Figure 10: Comparing performances of General Motors and Toyota (Womack et al 1990).

Starting in the 1950's Toyota began following the principle of **Kaizen** (1B) through inbuilt company-wide quality control also called **Jidoka** (1B), symbolizing a very different path in contrary to what took place at the western mass-assembly lines:

"The first step was to group workers into teams with a team leader rather than a foreman (..) Ohno next gave the team the job of housekeeping, minor tool repair, and quality checking. Finally, as the last step, after the teams were running smoothly, he set time aside periodically for teams to suggest ways collectively to improve the process" (Womack et al., 1990, p. 88)

There were not assigned any quality-control personnel; instead, the quality of the production was everyone's responsibility[4] (Karlsson & Åhlström, 1996). The workers at the assembly line were in fact encouraged to stop the line immediately, in an attempt of not pass on any

[4] For further insights into the early conceptualization on Lean see figure 20 in appendix 7.2.2.

defects further in the system, in order to pursue a culture of accountability in which everyone would actively pursue quality excellence through aiming for **Zero Defects** (1C).

In practice, these hands-on worker empowering initiatives leading to continuous process improvements, were aimed at eliminating the **3 M's** (1D) – being **Muda** (waste), **Mura** (unevenness) and **Muri** (overburden). Muda can be viewed upon as the lagging indicators, of which Muri and Mura are the indicators/scenarios leading to the visible Muda. To comprehend what **Muda** is it is reasonable to ask what value is, and whom it is for. In this case, Toyota set precedence in perceiving the value of their endeavours, in being ultimately defined by the end-customer or, for internal purposes, the person receiving the service (Womack et al., 1990). Thus all activities performed, consuming material or immaterial resources not adding direct value to the product/service in question, are considered an element of waste in the production. The Toyota philosophy does accordingly command a significant amount of effort and initiatives to seek to improve and remove these non-value adding elements (Karlsson & Åhlström, 1996). Whereas Muda has become very acknowledged and widespread in its logical reasoning, alarmingly few people understand how it is connected to **Mura,** the waste of unevenness or inconsistency. In its practical application, Mura means to balance the demand curve out[5], as a significant demand overload will establish Muda, for example, in waiting time at the production facility or to the holding cost of expensive and unnecessary inventory stock (Modig & Ahlstrom, 2012). On a higher level of abstraction, this pledge for consistency is also related to the introduction of management tools and organizational changes. This is an area where especially many western companies in the early years of the dissemination of Lean fell short in the consistent levelled introduction of Lean thinking. In this regard, it is relevant to mention **Kaikaku** (1A), which translates to radical improvement, and thereby is the opposite of Kaizen (Womack & Jones, 1996). Enforcing Kaikaku is truly needed in times of game-changing conditions, in which it can be an absolute necessity to leverage a given window of opportunity before it disappears. Nevertheless, enforcing Kaikaku arguably needs to be backed up immediately by a high degree of Kaizen in order to create stability for leveraging the continuous incremental improvement, and exploiting Lean's advantages in the current production design.

Finally, **Muri** means overburden, and can very often be a result of elements of Mura generating a stressful overload on employees and processes by, for example, not having sufficiently trained competences or falling short on having unreliable processes or bad communication channels (Modig & Ahlstrom, 2012). In the context of Toyota, Taiichi Ohno

[5] To level out the production to avoid Muda is also referred to as Heijunka.

specified how Muri and Mura lead to 7 categories of Muda being: transport, waiting time, motion, defectives, inventory as well as over- production and processing (Womack & Jones, 1996)**.** In turn, other categories for waste within manufacturing have been added to the categories, as later will be investigated more in-depth there has been developed specific categories of waste relevant to software development and IT organizations.

Providing the wrong good or service the right way is also considered Muda (Womack et al., 1990). It is however not clearly stated in the early studies on Lean and Toyota, to what extent you know what the customer wants; should the process integrate user-inputs, or have a development pipeline characterized solely as an engineering-push. To assess the waste throughout its entire relevant lifecycle, Toyota developed an end-to-end **Value-Stream** (1E) mapping tool, which encompassed all specific actions required from inventing to delivering the product to the consumer. This inter-organizational approach thereby deconstructed the silo way of thinking, as from a value perspective it was only concerned with the actual activities performed, and not what department was responsible for it.[6] In this way focusing on the flow of the value through the entire stream, means removing all impediments to continuous flow (Womack & Jones, 1996), by only letting the necessary process-steps and the activities that constitute value to the end-product remain. According to Modig & Ahlstrom (2012), the focus on unit flow constitutes in itself a new approach to efficiency, which is different from the traditional form of resource efficiency concerned with the maximum utilization of resources. In fact, the authors claim a focus on flow efficiency, defined as the sum of value-adding activities in relation to the throughput time, will eventually also lead to an increase in the traditional resource-based efficiency measure. Lean is therefore defined by its prioritizing on flow over resource efficiency by moving *to the right and up* in the efficiency matrix (Modig & Ahlstrom, 2012). This matrix is depicted in figure 21 located in appendix 7.2.2.

The theory of constraints (TOC) introduced by Eli Goldratt in 1984, can be argued to emphasize the same aspect of decreasing process friction, by seeking to identify and eliminate the bottlenecks in the process. This is done by enhancing the throughput time and capacity of the constraint, using levelling buffers and ropes to design the entire process to exploit the constraint.[7] A supporting principle to reduce waste, in this case eliminating *production buffers* is **Just-In-Time** (1F), which determines how each process ideally should only be provided

[6] A perspective similar to how activity-based costing and Lean accounting are later adopted to assess not the wastes, but the actual monetary cost of performing a service/product and further on to make profitability analysis on specific customer segments

[7] There exist several editions of the theory of applying the drum, buffer and rope on single or multiple constraints within processes to enhance throughput time, which will not directly be included in this study.

with the needed right part, at the right time, in the right quantity (Karlsson & Åhlström, 1996). Closely related to Just-in-Time is the **Pull** (1F) way of scheduling the delivery of material, which means a material will only be moved to the next stage when it has been triggered by internal or end-user demand. That is the opposite of having a push system, in which material are produced and stocked in inventory, waiting for orders to come in. Figure 22 illustrates the difference between push and pull in appendix 7.2.2.

Toyota used in turn a visualization of the production through the Kanban board, which depicted the current work in progress (WIP). The reduced batch size was composed by components, all of which had markers on them, delivering information to the Kanban board when the components of the batches were moved from one stage to another. The Kanban board will be examined further in chapters 4 and 5. For Toyota, this was only possible, due to a shared incentive structure through the entire supply chain, in which the idea was to build the dealer into the production system and the buyer into the product development process (Womack et al., 1990). As chain-supply as noted not will be within the boundaries of this study, the buyer/user-integration within the development process will thus be a key focus within the following section on Lean practices and schools.

3.3: Lean Product Development

The Lean principles effect on product development became already evident in retrospect in 1987 when Clark, Chew, Fujimoto, and Sheriff published their statistical findings from comparing 22 international automotive manufactures in their *Product Development in the World Auto Industry* publication (Hoppmann, Rebentisch, Dombrowski, & Zahn, 2011).

Product Development Performance by Regional Auto Industries, Mid-1980s

	Japanese Producers	*American Producers*	*European Volume Producers*	*European Specialist Producers*
Average Engineering Hours per New Car (millions)	1.7	3.1	2.9	3.1
Average Development Time per New Car (in months)	46.2	60.4	57.3	59.9
Number of Employees in Project Team	485	903	904	

Figure 11: Geographical differences in performances (Womack et al., 1990).

This data illustrates how the Japanese producers were extremely superior on every indicator to the rest of the world. The authors found evidence that this performance was due to 1) a strong involvement of suppliers in the design process 2) lead by the authority and

commitment of a heavy-weight project manager 3) through multifunctional teams 4) operating in reoccurring problem-solving cycles (Hoppmann et al., 2011). A working hypothesis was that the use of overlapping development stages was a main driver for productivity, which was later also indicated by the findings of (Womack et al., 1990). Through their findings at Toyota, they also added the practice of early and controlled communication to the list (Womack et al., 1990). Based on these findings, Lean can in this sense be viewed as enforcing an end-to-end flow, throughout the cycle of finding an initial market fit, optimizing the development and operational delivery. As straight-forward these building-blocks may seem, it has ever since been highlighted several times in the literature how the transition into a Lean product development process is supposedly tougher than to begin transforming a manufacturing assembly line into a Lean manufacturing process (León & Farris, 2011). This statement is namely inspired by one of the corner-sources in the area (Morgan & Liker, 2006), who were the first to translate the output of the examined Toyota practices into generic elements applicable for development processes beyond the borders of manufacturing.

A **Lean development** (1H) process is very much embedded in the organization, and therefore requires wide changes within people, processes and technology & tools[8]. These 3 categories have thus become dominant perspectives for integrating Lean and product development (León & Farris, 2011). It is important to understand how all these elements are causally connected in a coherent system, as they interact and overlap while being independent and working together. Former Toyota Vice-Chairman Fujio Cho explains:

(..) *What is important is integrating all the elements together into a system. It must be practiced every day in a very consistent manner - not in spurts.*
(Morgan & Liker, 2006 p. 9).

The starting point for this consistency, is arguably to have a good process encapsulating a lot of the previously mentioned Lean principles, as for example having established customer-defined value to separate value added from waste (Morgan & Liker, 2006). This matter needs to be addressed in particularly in the front-loading of the development process, in which alternative solutions should be thoroughly explored and assessed, as this is where there is maximum space for new ideas, functionalities and design. When the value and according waste are defined as well as a state of *design stability* is reached, a levelled product development flow has to be generated through standardization. The idea is to reduce variation

[8] For elaboration of the framework see figure 23 in appendix 7.2.2.

in order to create flexibility and predict outcomes (Morgan & Liker, 2006). As it is a challenge to reduce variation, whilst still preserving the creative nerve, it is crucial to standardize and structure low-level tasks through the usage of checklists, while heavy knowledge workers operate as *standardized specialists* to decrease dependency and generate flow flexibility. As the people are the ones providing the intelligence and energy, they constitute the culture through a shared language and ideally also with congruence in symbols, beliefs and values. In an organizational context, the silo department thinking and sporadic involvement in projects do not benefit from these cultural advantages of process-ownership. Having an assigned chief engineer, who follows the development from start to finish enables the human nerve and establishes both an element of a final authority and commitment through the entire process (Morgan & Liker, 2006).

When it comes to the correlation between the building blocks, a bullet point is to customize and adopt the technology & tools by acknowledging it is subordinated to fit the people and process. Technology is the one element that in it self can most easily be replicated. It has, however, a lot of greater value when it is suited the already highly optimized context of processes and organized skilled people (Morgan & Liker, 2006). To enhance a sustainable and competitive advantage, technology needs to fit and improve the processes and the performance of the people. In fact, the buzzword of alignment through simple visual communication and shared goals, are argued to enable joint problem solving in the development team and throughout the organization.

Moreover Brown (2007) and Schuh, Lenders and Hieber (2008) have progressed in adding cross-project knowledge transfer and rapid prototyping and testing to Morgan & Liker's elements of organizing Lean development (Hoppmann et al., 2011). Cross-project knowledge transfer builds upon organizational learning, and the art of disseminating knowledge throughout the organization, which builds upon the rotation of standardized-specialists who rotate among the projects. Nevertheless, it is obvious that time urgency and different project types are factors that can limit this exchange. Furthermore this understanding of Lean development seems to be designed to very large and plan-driven entities. The rapid prototyping is an extension of the usage of front-loading and in turn a way to determinate the value from the waste, as it enables other ranges of direct customer inputs, and is very much in line with what the Lean Start-up method prescribes.

3.3.1: Identifying value through customer development

Steven Blank was the first to define the customer development methodology in 2005 in his

book *The four steps to the Epiphany* (Mueller & Thoring, 2012). This basic idea is to enrich the product development phases, with an embedded or simultaneous running process of finding and understanding customers through actively seeking observations or direct feedback from them. The *customer development* was later adopted by the Lean Start-up movement, in order to use it as a systematic approach to craft new entities in an adaptive user-centric fashion (Ries, 2008). Eric Ries, the author of the Lean Start-up, defined a start-up as *"A human institution designed to create new products and services under conditions of extreme uncertainty"* (Ries, 2008 p. 8).

As much as new established entities fall under this umbrella, it is my impression that literature on the explorative side of organizational ambidexterity, and/or spin-offs[9] highlights this is as the case for a new established entity within or outside an existing company (Mueller & Thoring, 2012). There will therefore inevitably be some compatibility conflicts in fully uniting the Lean Start-up within very standardized product development systems as the ones depicted by Morgan & Liker. Nevertheless, the Lean Start-up approach's contribution in defining customer-value, very much seems to be the missing link in how to be absolute sure about, what your customer values. Operating through shorter Kaizen iterations, hopefully with credit to the 3 quality gurus, the goal is to follow the build-measure-learning cycle, in order to steer the development process to solve a specific problem or solution (Ries, 2008).

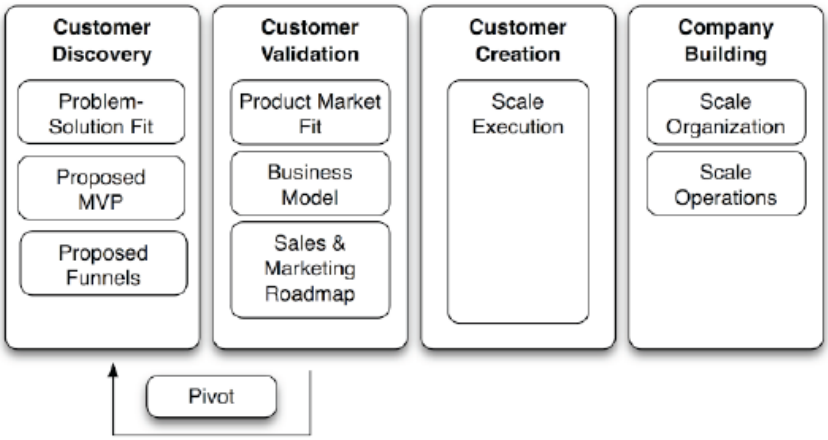


Figure 12: The Lean Start-up process of Customer Development (Mueller & Thoring, 2012)[10]

The **Customer Discovery (**1I) portrays how to detect and test reliable customer value propositions, through the creation of **MVPs** (1J), which are Minimum Viable Products, equal

[9] See for example (O'Reilly and Tushman 2013) or (Christensen 2000)
[10] As the Customer Creation & Company building are assed to be more concerned with the organizational aspects of scaling a start-up, it will not be examined further within this study.

to the rapid prototyping and testing previously mentioned. The Lean Start-up does however go further than the previous literature in its specification of customer co-creation through the Voice-of-Customer, and other crowd sourced open innovation methods.[11] The pivot relation between the customer validations gives us a first glimpse on agile development iterations, by somewhat extending the front-loading phase to repeat itself until the correct match between problem/solution and market/business model fit is there. Whether this is different from the front-load analysed by Morgan & Liker is difficult to determine, as it is explicitly stated that the front-load needs the capability to explore new solutions through a maximum design space, which could easily imply there in turn would exist process iterations before a design stability is reached. Nevertheless the highly industrial and standardized process proposed by Morgan & Liker, might be the reason for the prejudices a lot of software developers has towards Lean development. Whether this design stability will exists early, later or never in the process within Lean Software Development, is in my opinion even harder to answer. The following section will address this issue.

3.4: The emergence of Lean Software Development

In a recent comprehensive literature-review and content analysis on the dissemination of Lean, it is identified as increasingly being applied to IT organizations, mainly within application development and maintenance (Kobus & Westner, 2015). While the majority of the literature in the field is still related to IT within manufacturing, it is thus emerging within service functions, which are stated to be characterized by a higher degree of complexity and novelty (Kobus & Westner, 2015). Lane, Fitzgerald, and Ågerfalk (2012) studied Lean application development through the interdisciplinary lens of product development and operations- and project management, and found its existence to be more of a management philosophy rather than a specific methodology (Kobus & Westner, 2015). While this is very much in line with Lean in manufacturing (Modig & Ahlstrom, 2012), the scope of the philosophy in regards to software development seems even more uncertain due to a low degree of theory and empiric-based research built upon reference theories (Kobus & Westner, 2015).

Bob Charette in a series of articles in the mid 90's defined Lean Software Development as the operational piece of **Risk Entrepreneurship (1I)**, which is a three-levelled approach generating a change tolerant business. This framework can be found in figure 25 in appendix 7.2.2. The rationale is by exercising risk leadership, competitive advantage is stimulated in

[11] An overview of these methodologies is depicted in figure 24 located in appendix 7.2.2.

taking opportunity of change. For highly disruptive industries, the ability to maintain an efficient operation is looked upon as the key to simultaneously create and respond to change (Highsmith, 2002; Wang et al., 2012). The early conceptualization of Lean development was in that sense very much an abstraction of an operation strategy, and not explicitly prescriptive in terms of the employment of practices and tools.

In 2003 Tom & Marry Poppendieck, published their book Lean Software Development – An Agile Toolkit. In its very origin, the phrase was therefore highly inspired and builds to align, the Agile software movement's practices, within the principles of lean product development. The Poppendieck's approach to integrating Lean within software development (Wang et al., 2012), has later been characterized as the **Comprehensive School** (1J)[12]

3.4.1: The Comprehensive School

The name of the Comphrehensive School comes from its emphasis on transforming the essence of the full Lean methodology into a state of which it adds value in a software development setting. To elaborate, the Poppendieck's Lean principles applied to software development emerged as 7 principles in 2003[13], which prescribe to: 1) eliminate waste, 2) amplify learning, 3) decide as late as possible, 4) deliver as fast as possible, 5) empower the team, 6) build integrity in, and finally to 7) see the whole.

First and foremost the meaning of elimination of waste, very much follows the rationale of Tachi Ohno's mantra of "*He without bad habits has seven* ", to illustrate how "*even if you think there's no waste you will find at least seven types*" (Power & Conboy, 2014 p. 204). Covering everything that doesn't add value to the end-customer, the Poppendiecks' identified the following difference within waste in classic manufacturing and software development:

[12] Larman & Vodde 2009 & Leffinwell 2009 is also considered in this group, in which they more outspoken than the Poppendick's has translated the 14 principles of Lean from the *Lean production house* into software development.

[13] These were later redefined to 7 similar principles in 2007 and finally in 2014 were stated as 8 principles of a 'Lean mindset', which thus are not as specific to software development as the original 7 principles. The newest principles can be found on www.poppendieck.com

The Seven Wastes of Manufacturing	The Seven Wastes of Software Development
Inventory	Partially Done Work
Extra Processing	Extra Processes
Overproduction	Extra Features
Transportation	Task Switching
Waiting	Waiting
Motion	Motion
Defects	Defects

Figure 13: Waste applied to Software Development (Poppendieck & Poppendieck, 2003).

The principles of elimination of waste have since been one of the main topics of interest within Lean software development (Power & Conboy, 2014), and the word waste itself has been found to be a very emotive topic within teams and organizations. This is due to the fragmentation and negative implications it has on the people performing activities, which from a holistic systems perspective falls into the category of waste. Therefore it has become dominant within the literature to consider a focus on flow, rather than a focus on waste elimination. This is proves to be a better catalyst for continuous improvement within knowledge work activities such as software development[14] (Power & Conboy, 2014). Therefore waste has been rephrased as **impediments to flow** (3A), which from my perspective is a combination of Lean and Agile principles, and have led to the extra sources of *unmet human potential*, *context switching* and *failure demand* (Power & Conboy, 2014).

This evolvement is very symptomatic with the evolution of the categories of waste within manufacturing, in which the unmet human potential, for example, accounts for not utilizing the talent capabilities available. The context switching means losing the flow of focus by being involved in too many activities. Failure demand simply implicates putting the wrong demand on the development process, which is similar to Mura, and in essence, not adding value by solving the right problem/pain for the customer.

Back to the 7 principles of the Poppendick's, the amplification of learning should take place in iterations. This means within software development to constantly generate value, through a continuous quality and fitness validations, whereas iterations within production is a strictly source of waste (Poppendieck & Poppendieck, 2003). It can therefore be compared to developing a recipe and to produce by following that recipe, in which the amplification of learning needs to take place through iterations, similar to the rationale provided by Lean startup. This element seems very much interconnected with the empowerment of the team building in integrity and seeing the whole, as empowered teams provides the motivation and

[14] For further information on these categories see figure 26 in appendix 7.2.2.

integrity to provide the best solution for the development process in its entirety. The process ownership, therefore provide an incentive not to engage in individual gaming of the system, but to enforce collaboration leading to real life impacts, as a results of the amplified learning iterations. Finally, deciding as late as possible and delivering as fast as possible mean that you cannot see the whole of the end-product before you have engaged iterations and received valuable internal or external feedback on the output. To deliver as fast as possible means to engage in the iterations and test hypotheses as fast as possible, which also implies to fail as fast as possible. This will make sure you do not consume enormous resources on elements that in the end will never earn value for the customer.

It is also worth mentioning Lean Six Sigma[15], which in the recent years has also received significant attention in larger software companies, due to the efficiency and quality benefits that standardization thus may generate. Inspired by the already mentioned Deming's Circle, the goal is to standardize processes, in that sense breaking away from the above mentioned Agile inspired development discourse. Figure 27 illustrates in appendix 7.2.2 different situational applications of Lean and Six Sigma within software development (Pillai, Pundir, & Ganapathy, 2012).

How some of these key elements in the Comprehensive school are combined with Agile practices within the Selective school will be investigated, after the foundations of the Agile movement and *Chaordic* software development have been examined in the following section.

3.5: Agile Foundation and Principles

In this section, I will attempt to back up some of the comments that have already been made on Agile software development. Although the progress will follow the previous Lean section, it will nevertheless more directly emphasize the empirical evidence that exists on the Agile principles and practices.

3.5.1: The Agile Movement

On the 11th and 13th of February in 2001, 17 well-known contributing thought leaders within the software community met at the Lodge at the Snowbird ski resort in the Wasatch

[15] Six Sigma is a well-structured data-driven approach originated to improve manufacturing processes and to eliminate defects. Six sigma was originally a collection of practices such as Quality Control, TQM, and Zero Defects (Pillai et al., 2012). For elaboration see figure 35 in appendix 7.2.2.

Mountains of Utah. The purpose was to talk, ski, and relax as well as to find a common ground for the future direction of software development (Highsmith 2002: 11) The outcome of the stay was the 12 principles of the Agilemanifesto.org, which gave birth to the rise of the Agile software development movement. In summary the fundamentals of the Agile Manifesto's principles can be summarized as follow:

- **Individuals and interactions** (2A) over processes and tools
- **Working software** (2B) over comprehensive documentation
- **Customer collaboration** (2C) over contract negotiation
- **Responding to change** (2D) over following a plan

The authors stated that while there was value in the items to the right, they valued the items on the left side more[16] (Cockburn, 2002; Highsmith, 2002). The meeting was in that regard a manifestation of the imbalance of what was considered most crucial for successful development processes. It was in a sense a reaction towards traditional software development[17], and its prioritization of processes, rather than what fostered the best software solutions. While both aspects were important, the goal was to restore the proportions between the left and right side. One powerful argument to illustrate their correlation is the logical reasoning, that it is not possible to have a successful project by delivering documentation without working software, but it is however possible to deliver working software without any documentation (Cockburn, 2002; Highsmith, 2002).

3.5.2: The evolution of Chaordic software development

The information age economy is characterized by a great deal of speed and change. Therefore to innovate better and faster, by responding to new technology and customer requirements, is the pace that will create change for competitors. If not you are the one driving the change, you're are doomed to be changed by your competitors, and have to adopt survival strategies on the premises of their pace (Highsmith, 2002). While the Internet itself qualifies as a main driver for this evolution, it is the acceleration and emergence of multiple types of changes that are the symptoms. In fact, this acknowledgement is the reasoning for the name of the movement, in which one common definition is:

"*Agility is the ability to both create and respond to change in order to profit in a turbulent*

[16] In 2009 additional extensions were made to these 4 principles in the software craftsmanship manifesto, located at http://manifesto.softwarecraftsmanship.org

[17] For a structured overview of the main differences on the two see figure 28 in appendix 7.2.2.

business environment" (Highsmith 2002, Cockburn, 2002, p. 29).

While this might have been ancient industry dynamics, the information economy has very rapidly changed the sufficient speed for setting the pace. A Chaordic perspective acknowledges this and assumes, that every organization has properties of chaos and order, and should accordingly use both properties in their way to manage project teams and organizations. This Chaordic perspective draws on the philosophy of complex adaptive human systems (Power & Conboy, 2014). In this perspective, complex solutions are assumed to require decentralized independent agents, who are interacting through **self-organizing** (2D) teams. In which they are guided by a common understanding of generative rules (Cockburn, 2002). It is noteworthy it is within self-organizing teams that the epistemologies of Lean and Agile have been found to be most compatible (Hines, Holweg, Rich, Browaeys, & Fisser, 2012). Moreover several studies highlights, the parallel of the emergence of Lean production and Agile software development, as both simultaneously seek to improve quality, lower costs, and increase speed to market with the means of focusing on people (Highsmith, 2002; Hines et al., 2012; Wang et al., 2012).

In connection to Lean Development (Morgan & Liker, 2006), the starting point is supposedly in the processes. The agile movement's focal point, on the other hand, is that all individuals are unique, and the processes should accordingly be forged to the uniqueness of individuals and teams. Hence the key is not blindly following management methodologies to construct processes that do not naturally fit the flow of the individuals. Herein lies the assumption that no people are average components, and it is not processes, but people, that turn people into star performers. This can thus only happen if they are valued, trusted and supported (Highsmith, 2002). To make a general distinction between processes and skills requires processes to enhance the skills, and not restrict them through a formal process-centric approach. In accordance, as the mantra states *working software through people* also implies that the project team are to determine the level of documentation for working software. It also has to be acknowledged that a great deal of the tacit knowledge constituting the skills, in many cases cannot be shared by tangible documentation (Cockburn, 2002).

In regards to contract negotiation, be it an internal project charter or an external legal contract it is assumed to "(..) *Encourage the addition of buffers*", which "(..) *makes projects take even longer, drives up costs, and reduces responsiveness to change*" (Highsmith, 2002 p. xviii). In line with the Lean principles and methodology of Lean Start-up (Ries, 2008), the relationship between customer, suppliers and developers should ideally be a collaborative arrangement to find the best suitable solution.

While it is often mentioned Lean Start-up is built upon the Agile movements approach to *customer development*, it is also very evident how the customer-delivery principles within the Agile movement is based upon Kaizen, by reducing activities not relevant for working-software. Furthermore the incremental nature of continuous improvement, in contrary to the radical changes of Kaikaku, is clearly the same reasoning behind the agile principle of **Continous Integration** (2E). Continous integration is typically a testing-activity, in which a new code might go through instant unit testing right after or meanwhile it is made, in order to check for errors. This breaks the software development into very short cycles, as a logical way of responding to a Chaordic environment.

There are clearly some rather legal advantages in terms of risk, in having established the development transaction on contractual grounds, by setting up boundary restrictions to the scoop and timeframe of the involved parties. It thus has to be realized that this requires the agreement on a plan, which in turn is the opposite of being capable of adjusting to change. As depicted in figure 14, following a plan is valuable, but stated to be less valuable than having working software for the user as a result of an agile development process.

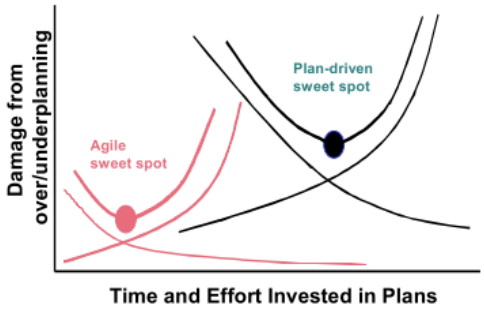


Figure 14: Finding the appropriate 'sweet spot' of planning (Cockburn, 2002)

Another argument for this rationale is demand estimation, as robust planning and contracts requires the forecasting of the user-demand for the entire process, and thus the customer needs to have a strong knowledge-domain in order to formulate a comprehensive *master-plan* of needs (Highsmith, 2002). For the same reason contracts might therefore in turn be somewhat subscription or milestone[18] based, in order to find a compromise for parties, which have a hard time relying solely on trust.

3.5.3: The empirical evidence on Agile practices

[18] Escrow.com is an example of such an intermediary facilitating milestone payments. In other realms of the Internet the *no cure no pay* model is also becoming widespread on for example crowd sourced development sites.

There is a variety of methodological practices, within the Agile software ecosystem[19], of which the majority have their core roots in the authors of the Agile Manifesto (Highsmith, 2002). In fact several of these approaches are based on lightweight teams (Dingsøyr, Nerur, Balijepally, & Moe, 2012), user-centered requirements planning (Salah, Paige, & Cairns, 2014), shared decision-making (Moe, Aurum, & Dybå, 2012) and mental modes of collaboration (Yu & Petter, 2014) coordinated through co-located communication (Pikkarainen, Haikara, Salo, Abrahamsson, & Still, 2008) started to appear during the 90's. Today one of the most recent surveys estimates a world-wide adoption level above fifty % of single or hybrid elements of agile practices within software development processes (Strode, 2015).

Agile practices have thus in its success also evolved into being applied as a heavyweight methodology in large enterprises, especially accompanied by elements of Lean development (Pernstål, Feldt, & Gorschek, 2013). Also found to prove useful within research based scientific software development (Sletholt, Hannay, Pfahl, Benestad, & Langtangen, 2011).

The nature of communication has with the rise of new internet-based applications, moderated the time/space dichotomy into making Agile practices used within distributed software engineering (Rizvi, Bagheri, & Gasevic, 2015). In that way, the foundation of Agile software is changed to that based on physical co-location, by enabling co-location in virtual applications such as Gitter or Slack. This has also modified the traditional co-location into being a hybrid of virtual co-location. In 2015 a multiple-case study on coordination and collaboration on Agile projects (Strode, 2015) illustrated how 76 % of all dependencies across the projects were knowledge dependencies, exceeding the dependencies of processes and resources 3 to 4 times. This illustrates how the founding principles within the Agile movement today still value tacit knowledge within individuals over explicit process knowledge.

A review on user-centric development revealed the usability methods are thus often used too late in the development process (Brhel et al., 2015). Although **Scrum (**2F**)** improve the understanding of actual user needs, it is found harder for teams to translate this knowledge into a specific user-requirements goal. To this end the most frequent user-centric practices are found to be fast prototyping, individual inquiry, formal tests, and heuristic evaluations (Brhel et al., 2015). It has to be noted that many of the self-organizing elements of Scrum dates all the way back to (Nonaka & Takeuchi, 1986), but it was not framed as a distinct practice

[19] Figure 29 offers an introduction to the practices in appendix 7.2.2. The practices, which will be analyzed in depth within this study will although, be presented gradually during the study.

within software before co-author of the Agile Manifesto Ken Schwaber, in 2002 published his book on the topic. In this corner book on the Agile development ecosystem (Highsmith, 2002) interviews are made with Ken, who offered this definition:

"Scrum provides a project management framework that focuses development into 30.day sprint cycles in which a specified set of Backlog features are delivered. The core practice in Scrum is the use of daily 15-minutes team meetings for coordination and integration" (p.xxxii).

Scrum has since become so popular that there are now several different Scrum certificates in the market. The community views on how to use Scrum on a practical level will be further examined in chapter 4, and in relation to two of the cases in chapter 5.

In relation to this, a systematic review from Norway synthesizing the empirical studies within the Agile ecosystem found that the tendency at that time was with rare exceptions, only **Extreme Programming (**2G**)** had been studied (Dybå & Dingsøyr, 2008). Further the authors stated:

" (..) In our opinion, management-oriented approaches, such as Scrum, are clearly the most under-researched compared to their popularity in industry" (p. 852).

This is followed by the argumentation that future research studies should be designed to study Agile approaches that are popular within the context of the industry (Dybå & Dingsøyr, 2008). In a survey within the Finnish software industry, Rodríguez, Markkula, Oivo, & Turula, 2012 also found Scrum very significantly being the most widespread Agile practice within their 420 responses. Followed by practices such as Extreme Programming (XP), **Feature-Driven Development** (2H) and Kanban[20].

This indicates that Scrum and XP are the practices, which both scholars and practitioners have adopted most frequently. Providing evidence that the empirical studies of methods such as test-driven development, feature-driven development, Lean and Kanban are very vague if existing (Inayat, Salim, Marczak, Daneva, & Shamshirband, 2015). Also highlighting it to be

[20] Other low representative approaches in the study also included: Crystal methods, Dynamic Software Development (DSD), Adaptive Software Development (KSD), Rapid Application Development (RAD) Test-driven development (TTD) & Bob Charette's Lean Development.

necessary to assess their response to the agile way of managing requirements in regards to uncovering the impact of Lean and Kanban practices (Inayat et al., 2015).

Whereas Kanban is by some claimed to be a practice, it is also described as being a tool within Lean. This weak differentiation between Agile, Kanban & Lean has lead researchers to include both understandings of Kanban within their research (Kupiainen, Mäntylä, & Itkonen, 2015). While these arguments are surely very valid, I have for a matter of simplicity decided to categorize the Kanban as a practice within Lean. Before this will be examined in chapter 4 and 5, the literature that actively combines the concepts of Lean and Agile will be analysed next.

3.6: Combining Lean and Agile within Software Development

As the origin and inter-complexity among the concepts have now been examined, it is now appropriate to introduce the selected literature combining the concepts. This final part of the literature review will be divided into a section on Leagile software development, as well as the practical oriented Selective School. Lastly the research on performance metrics will be examined before the conclusion of the review. First, I would like to establish a shared understanding in regard to what constitutes a software development process[21], based on what is mentioned on agileweboperations.com, shown in figure 15. The figure shows a synthesis between Lean and Agile in relation to DevOps.

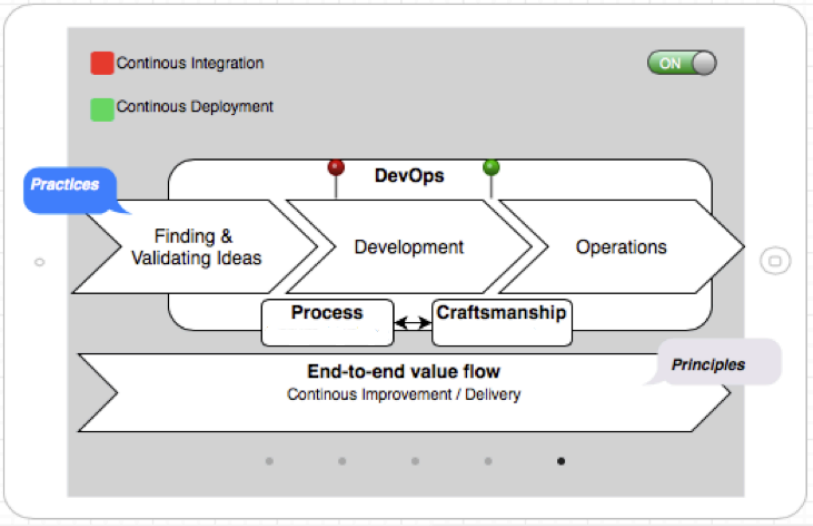


Figure 15: The software development process. Own creation.

[21] This figure is far from offering a full-picture into the dynamics between the domains, in which iterations will cause the process not to be linear in most cases.

As many of the concepts on the illustration will be discussed throughout the rest of this and the following chapters, it is for now sufficient to highlight the basic point that ideas eventually goes from validation into development and finally into the production/operation. Typically it would be continuously tested and integrated, and eventually deployed. The process and craftsmanship boxes illustrate how the craftsmanship is hands-on engineering-based aspect, whereas the process category is concerned with the planning, visualization and optimization of the entire development process.

3.6.1: Leagile software development

Lean within software was commonly viewed upon, as *yet another* Agile practice (Dybå & Dingsøyr, 2008; Highsmith, 2002). However, some claim Lean provides the theoretical foundations for Agile practices (Poppendieck & Poppendieck, 2010). Finally others argue, that adding Lean to Agile is a progression from moving planning to the project/team level up to an organizational level, which Agile practices are not suitable for (Smits, 2007).

This increasing interest and disagreement of how to frame the combination of the concepts has lead (Wang, Conboy & Cawley 2012) to categorize different degrees of such combinations as **Leagile** (3E) software development. This contraction is clearly inspired by the phrase *leagility*, which has been applied on combining the concepts within chain-supplies (Wang et al., 2012).

In a study based on 30 experience reports, six different archetypes of combining Lean and Agile practices were presented (Wang et al., 2012). One non-purposeful combination and five purposeful combinations were found. Among the latter five combinations, two were approaches as a progression on a Lean practice, and two as a progression on an Agile practice. The last was having the two concepts synchronized by having teams working in parallel[22] (Wang et al., 2012). This study revealed how the division within the Comprehensive School and the Selective School in practice also exists, as there are various combinations of how to combine and coin Leagile. Finally, the researchers pledged others studies to:

"*Provide operational guidance to help developers (i) map the various potential leagile processs to their own context and (ii) implement the selected process combination"* (Wang et al., 2012 p. 22).

[22] The entire framework suggested can be seen in figure 30 in appendix 7.2.2.

Despite, and perhaps due to the definitorial fragmentation, there is a strong need for a better understanding of the potential applications and benefits of applying leagile practices within software development, and especially study how they are working together in mature-development teams, where they have already been implemented (Dybå & Dingsøyr, 2008).

3.6.2: The Selective School

Building upon some of the same fundamental principles as its comprehensive ancestor, the **Selective School** (3F) only encompass elements on a *pick-and-choose* basis, determined by the organizational context and dissemination and maturity of other practices. The main contributors (Ladas 2009, Anderson 2010, Kniberg & Skarin 2010 & Scotland 2012), focused on the Kanban method which has grown to a full-scale methodology, of which there even exists several variations (Wang et al., 2012). A systematic review on the Kanban approach including 34 studies conducted in 2014 (Al-Baik & Miller, 2014), found Kanban to be the most frequently used Lean practice within product development. Further they uncovered 46 % of the studies used Kanban as an efficient tool to visualize and prioritize the development cycle. Moreover 32 % of the studies reporting Kanban being leveraged as a means to leading organizational change as well as facilitating cross-functional teamwork (Al-Baik & Miller, 2014)[23]. Kanban's association to the Agile methodology is thus not to debate, which the abbreviation of Scrum and Kanban referred to as **Scrumban** (3G) also highlights. In fact the combination of these two methods has recently received a strong attention in academic circles and among professionals (Khan 2014, Reddy 2016). In the book Scrumban from January 2016, the author states:

"It emphasizes applying Kanban systems within a Scrum context, and layering the Kanban method alongside Scrum as a vehicle for evolutionary change" (Reddy, 2016). As will be analysed in the chapters of 4 and 5, Scrumban it is in that sense a way of using the prescriptive roles of Scrum, and a way utilizing either sprint or continuous flow when appropriate. As previously mentioned the great implications of change-management will not be included in this thesis, but rather the scope will be the potential synergy effect of using Scrum and Kanban simultaneously. Finally the researchers highlighted that:

"The Lean approach is more than the Kanban method and it has been proposed that the,

[23] The authors further highlighting that there exist 4 main categories: 1) Kanban as an element of Lean (beyond agile), 2) Kanban as an agile methodology, 3) Hybrid of Kanban and Scrum, and 4) Explanatory or change-driven Kanban (Al-Baik & Miller, 2014)

Kanban approach should use more approaches from the Lean principle so as to take the full advantage of the Lean principles in the software development"
(Al-Baik & Miller, 2014. p.1890).

While the Kanban has led the way to the Selective School, it suggests that there are several more elements of combining Lean and Agile that perhaps are already used, but not widely understood. Another notable finding in this regard, which however has not been very well documented elsewhere, is the application of the Lean principle Jidoka within software applications and IT-architecture (Danovaro, Janes, & Succi, 2008). This might sound very abstract, but from my point of view it seems very intuitive with the range of semi or full automatic processes and tools that are now shaping the development process. This aspect will also be discussed and exemplified in chapter 5.

Finally the new rising star in the development environment **DevOps** (3H)[24] has also been indicated to be a way of leveraging the purposeful combination of elements of Lean and Agile principles (Floris, Amrit, & Daneva, 2014). As Lean in many larger organizations are typically associated with the production area, and Agile with the development realm, it is no wonder how to break-down these silos might also contribute to some degree of assimilation among their respective culture in this regard. In other words, DevOps can be argued as using the principles of **Continous Delivery** (3C) and **Continuous Deployment** (3D)[25] as a way of creating a necessary shared process-ownership among the employees, in order to facilitate an end-to-end process of **Continuous Improvement** (3B). Supposedly it can hereby fill the implementation gap, through direct involvement of relevant parties, and in that way make sure to follow the Agile iterations *all the way through*, by seeking to offer the best collaborative conditions for testing and validating. The aspects in which many organizations might be a wall of silo thinking, exemplified by internal bureaucracy such as ticket systems. In other words systems, which can create barriers for collaboration and process ownership. It has to be noted these dynamics between the principles of Agile and Lean has not yet been investigated and/or documented[26].

[24] DevOps is an abbreviation of Development and Operations and is a philosophy of collaboration between the two traditional separate departments. It resembles a vision of creating a culture of collaboration, through a *pick-and-choose* attitude towards proven practices (Swartout, 2012)
[25] CI means to merge all developer working copies to the same 'mainline', having a common base for the future progress on the project. It is stated by XP to integrate as many as 10 times a day. Deployment is the step after delivery, it is the consistent deploying of code to production as the new developed features are completed. Often the building developer is responsible for following the code all the way until deployment.
[26] Figure 31 in appendix 7.2.2 show how Kai Petersons found links between all the principles, except for the end-to-end flow. This was importantly in 2010, and thereby before the rise of DevOps.

3.6.3: Literature on performance metrics

Although there has not been found any significant research conducted within the scope of explicitly combining Lean and Agile tools and activities, there have been instances of insights on performance metrics. On this matter I would like to recap the distinction between practices as: *How we should perform* and the definition of performance metrics as: *How have we performed and how can this be improved.* The point being that *should* in the first instance is defined as the normative performance evaluation criteria of which the later performance and its improvement should be compared to. It is therefore an absolute must to have congruence between the practice and the way of evaluating.

A recent literature review on the relation between metrics and the usage of Lean and Agile (Kupiainen et al., 2015) offers such insights into the occurrence of metrics and their relation to the practices/methodologies. From a study of 30 selected primary studies, the measures of 1) velocity 2) effort estimate 3) defect count 4) work-in progress and 5) customer satisfaction are found to be the ones most frequently occurring (Kupiainen et al., 2015). These results are somewhat in line with earlier study on performance metrics limited to Agile teams, with the exception of burndown and burnup charts which were also on the list (Javdani, Zulzalil, Ghani, Sultan, & Parizi, 2013). For simplicity these will not be included in this section.

The velocity and effort estimate measures have both links to the Lean, XP and Scrum practices/methods. While effort estimate is a planning activity, velocity is sometimes argued to be a productivity indicator. Without going further into this debate, velocity in this study will be defined as *the number of completed user stories in any given iteration* (Javdani et al., 2013). It can therefore be used typically for estimating the remaining time to the end of a project, as well as indicate who have been most diligent within the development process. This is obviously only the case if the coding commits are transparent, as well as if the team agrees on the nominal weight of the story point or which ever sub-measure each subtask has. This aspect will be analysed further in depth in chapter 4 and 5.

The defect count and customer satisfaction can in their core be argued to be measures for respectively pre-release quality and post-release quality. However the study revealed that both are actively deployed throughout all development activities, for the purpose of employee motivation and identification and fixing of customer and process problems (Kupiainen et al., 2015). Defect counts has from my perspective typically been associated with the standardized nature of methods such as Six-Sigma. Or in other words in situations in which the output of the process can be somewhat expected to fulfil other criteria. As Kanban in fact visualize the

total activities, work-in-progress is in that sense also relevant as measurements for lead and cycle time throughout the entire development evolution. This way of measurement seems very much similar to the throughput time and its application of the theory of constraints described within the Lean section. Nevertheless, work-in-progress might at the same time make the impediments to flow visible, by depicting the bottlenecks on the most explorative tiny projects. The measure can from my understanding therefore have a double role in both being an inputs to a larger calculation of throughput, or an attribute which forces the team to look at the on-going activities. In other words a way to make sure to address the waste appearing as a result of to much context switching. This aspect is of course heavily influenced by whether this is something the team can choose, for example depending on if and how the team is utilizing sprints or a working according got a continuous flow process. Before these elements are addressed in chapter 4, the conclusion will state the main findings as well as the accumulated recommendations for future studies.

3.7: Conclusion

As an output of the literature review, figure 32 in appendix 7.2.2, illustrates the findings of the foundation/principles and practices/schools within the distinctions of literature on Lean, Agile and Combined. It has to be noted that as with any categorization, figure 32 does not in its simplicity address the complexity of the reality. As highlighted throughout the review, the origin, principles and practices of Lean and Agile are extremely interconnected, which is why a lot of the elements in the figure could easily switch places. This chapter, illustrated how the rise of Agile methodologies, is based on craftsmanship, in response to the rigid planning-oriented concepts, originated from manufacturing. Where as the Lean development by (Morgan & Liker, 2006), through the elimination of process variability also have the same degree of standardization, it is thus through a people-centric process. The Lean start-up and the Lean software development by (Poppendieck & Poppendieck, 2003) where found to be directly inspired by the Agile movement. Moreover Scrum and Kanban has become so popular, so they in turn are often utilized as independent methodologies. Finally DevOps was hypothesized, as being the missing link for having an end-to-end flow, with the relevance for both Lean and Agile processes. While this is not the most straightforward output, I believe it illustrates how there exist disagreements on the conceptualization of something being either Lean or Agile. This is highly unproductive, and calls for a better understanding of what the concepts offers, in order for practitioners to pick and choose based on what might fit into their unique context in question. In this regard it is also noteworthy how little or none of the literature, with the exception of the findings in terms of performance metrics, which has

addressed the aspect of tools and activities. This aspect alone will be of high priority to uncover in the following chapters, along with the sum of the collected suggestions presented in the following section.

3.8.1: Empirical findings and suggestions for further research

What in contrary to this preliminary conclusion has been very clear, is the sum of suggestions for the state of the field and the needed further research:

- Lean and Agile Software development are both individually and especially combined nascent research areas (Al-Baik & Miller, 2014; Wang et al., 2012)
- Lean and Agile should be studied within mature development teams, and the scope should be some of the common elements among practitioners, which are clearly proportional underrepresented within academic studies (Dybå & Dingsøyr, 2008; Sjøberg et al., 2007)
- Scrum, Extreme Programming, Kanban and Feature-Driven Development have been found to be some of the most used practices (Rodríguez, Markkula, Oivo, & Turula, 2012).
- Scrum and Extreme-Programming are the practices which have been studied mostly in depth, with few conducted studies on Feature-Driven development and Kanban (Al-Baik & Miller, 2014)

With these good advice in hand, the following parts of the study will aim to progress in order to present new insights to these gaps within the literature.

CHAPTER 4: LINKEDIN CONTENT ANALYSIS

4.0: Overview

The LinkedIn search and content analysis was performed in order to indicate how the software community in turn utilizes Lean and Agile. Further the emphasis will be to comprehend how practices come into life, by studying the usage of tools, activities and performance metrics. Although this chapter will thereby mainly focus on the data derived from LinkedIn, points made by the community that supports or challenge statements in the literature review is included along the way. After a short introduction to the quantitative search findings within the selected LinkedIn group, this chapter will present the

categorization and findings from the 20 selected posts and their associated selected comments.

4.1: Search findings and categorization of posts

As mentioned in the LinkedIn methodology section, the search result from the 154 posts by 130 authors has seems to have been too small[27]. This is well illustrated with the no direct findings on practices such a XP, which is in contrary with previous surveys conducted findings (Rodríguez et al., 2012). With this said, the quality of the posts have although provided rich data to better understand how Lean and Agile practices, tools, activities and performance metrics are utilized. This is namely within the main categorizations listed in figure 33 in appendix 7.2.2. As is illustrated in figure 22, the 20 selected posts were divided into the four main categories of Scrum, Kanban, DevOps and Planning & Motivation.

4.2 Planning & motivation

This categorization will go into how the relationship between elements of planning, infrastructure and architecture unfolds. First of all the community thoughts on how to set the right conditions for motivation creativity and innovation will be briefly examined.
From the literature the planning sweet spot (Cockburn, 2002), and its interconnectivity with the aspects of human based attributes of inspiration and motivation were well documented (Strode, 2015). This also seems to be the opinion in the community, which describes how aspects of having an Agile team provided with autonomy, mastery, and purpose are commonly rooted as drivers for creativity and innovation (#1 Allen Holub). Through this craftsmanship autonomy, the question does in that sense comes down to, what constitutes the purpose or goal the individual/team has (#1 Chris Alexander). A way of creating such a culture is through dedication of a fixed amount of resources, for example 20 % outside the sprints to activities developers themselves chooses in what to do. Within this timeframe there can also be presented general problems, in which the members can come up with non-constrained open-ended solutions to solve (#1 Guy Maslen). A well-functioned cognitively diverse team provides in that sense the foundation for creativity. The right organizational culture and process does along with mentorship and coaching acts as a multiplier (#1 Brad Black). While this archetype of how an agile human-centric approach drives motivation and

[27] Another dimension to this discrepancy might be that XP was not included directly in the titles, tags or post description by the creator. It has nevertheless been included in some of the community responses. Further many aspects of XP is embedded within the posts and comments without explicit mentioning XP.

creativity, similar to (Pikkarainen et al., 2008), the community also pay a lot of thought into planning and documentation.

One example of this is roadmaps and product visions, which sets the boundaries for the development on a continuum ranging from specific to high abstractive. The stronger the detail and impact the roadmaps have, they are argued to ensemble a more traditional waterfall project management approach. Nevertheless some Agile editions of such a map may not include a timeline, but instead they are limited to express the puzzle of story pieces together constituting the entire epic story[28] (#14 Zijian Huang). In a greater time scope, it might also be valuable to deliver a technology roadmap, driving a series of isolated applications into a common technology platform. There should not be any specific timeline on such a map; hence the idea is to make sure that the new products releases or epic stories, fit into this overall technology strategy (#14 Guy Maslen). Infrastructure & architectural changes are essentially based on feature needs derived from user-cases, which in complex software development can possible evolve over time (#6 Eric Tucker).

Even though the very limited direct findings of Feature-driven-development and Lean start-up within the LinkedIn group, the above statements illustrates how the user-case and features plays a significant role for the initial mapping of the design of the architecture. The architecture is in that sense a natural part of a deliverable feature, so if the feature evolves the architecture might do the same. If the chosen architecture has implications that need to be addressed up front, the key is simplicity through the agreement on its minimum level, and then to refine it as the process move forwards (#6 Chris Alexander). In practice it might however be different, whether the team has the capability and access to make changes at that level. In contrary the team might need to test in safe mode or acquire cross-functional assistance from the infrastructure development team (#6 Eric Tucker).

The logic of the creation of a MVP from Lean Start-up (Ries, 2008), is in that sense integrated in the way architectural changes are conducted. How such a validation of the MVP takes place, will among other aspects be examined within the following section, after the basic dynamics of Scrum have been elaborated.

4.3: Scrum

In this section the community opinions on Scrum will be examined in order to understand the basic roles and how tasks and responsibilities are assigned. It will furthermore go through

[28] An epic story is a high-level usercase or feature, and in that sense a body of many of stories before it has been split.

how sprints, and retro perspectives are or should be conducted. Finally the community thoughts on the usage of performance metrics and contracts will be analysed.

4.3.1: Tasks and responsibilities within Scrum

Very much in line with the previous definition by Ken Schwaber (Highsmith, 2002), Scrum is stated to be about breaking the development project into cadences, also popularly called sprints. First and foremost a sprint is defined as a fixed and/or time-boxed activity (#2 Allen Holub). It is in other words a defined duration, in which the team has allocated specific development tasks, or what is often called *stories* to be completed within that period. Scrum is in that sense the opposite of operating through a continuous flow/pull system such as Kanban (2# Allen Holub). Scrum prescribes furthermore certain roles, as well as different process-steps, which would normally be the pillars of planning, daily-stand-ups, review and finally an evaluation through a retro perspective (Ajay Reddy, 2016)

The Scrum Master (SM) role is, clearly inspired by the underlying Lean principles, to remove the impediments to flow (Power & Conboy, 2014) for the team (#12 Tom Mellor). This means to assure everyone understands and uses Scrum effectively in their delivering of services to for example the Product Owner (PO) (#12 Paul Oldfield). Although the scrum masters attendance at the daily stand-up meetings might support this, the SM's presence could also be obsolete within a mature self-organizing team (#12 Padma Satyamurthy). So as Scrum is only a guiding and not a rigid process, it depends on the context, as well as whether the presence of the SM or PO removes the impediments to flow in their attendance (#12 Tom Mellor). The PO role is, perhaps in dialogue with the SM, the team or other stakeholders, to initiate the process with the formulation of a need. In that sense creating a demand to the development team in a form of an intangible vision or a more specific release plan/road map. Whether these responsibilities are shared or individual, depends on the dogma in the organizations, as well as the experience with Scrum the PO, SM and the team have (#15 Paul Oldfield). A SM should not have a classic project manager role of *focusing on deadlines*, instead the team should do their best to update stories at any time during sprints, and not end up in an unbalanced final rush (#15 David Johnson). A reoccurring and symptomatic opinion that illustrates the allergy a lot of the community has to remains of the pre-agile way of managing projects. A project-oriented waterfall approach that to thus some degree is the dogma in many organizations, which the case studies in chapter 5 will also illustrate. Another element of Scrum is continuous integration, which in turn requires the activity of formulating a *definition of done* (DoD), as well as a minimum acceptance criteria. The acceptance criteria are in this regard setting the boundary for the DoD, which in that way

defines the starting-point of inbuilt quality. This could for example be a common consensus of "*all code has been unit tested"* (#9 Chris Alexanders).

In this regard it is allegedly an advantage if it is the testers who build the acceptance test, which should exist before the process of writing the code begins. It is further claimed that there should not be any non-coding *monkey-testers* and therefore no passive testers, as there is no time for it on Agile teams (#20 Allen Holub). This point of view illustrates the dissatisfaction with non-coding members in regards to testing, but it is nevertheless not explicit stated how other levels of interface and customer testing could be done by employees who cannot code. In terms of software automation of testing to continuous-integrate, it has to be noted there can also be great value in exploratory testing, which automation software are not able to perform (#20 Janet Gregory). Testers would in some instances also wrap up different future scenarios, as well as decide and propose which regressions tests should be made, in collaboration with the PO. Finding work for a tester should therefore not be any challenge (#20 Carla Severi). It is possible to assign a specific quality role, to assure the product to be shippable throughout the different development stages. This can be a job for both individuals and/or an entire DevOps team. In case an individual performs the quality checks, the stronger the tendency the team will not posses a significant product or process ownership (#9 Michael Farag). This aspect highlights the Ford dilemma, in terms of having quality as a responsibility on a team vs. individual level. It although has to be noted how all team members in any case in Scrum, have a stake in the inbuilt quality level qua their participation on the planning and retro perspective meetings.

4.3.2: How to handle stories within sprints

In many cases the sprint does of course not turn out as initial planned. The appearance of incomplete user stories could for example happen due to a mismatch in story size and/or the sprint length, by which either breaking up stories or extending the sprint might be considered. The team has to discuss the occurrence in the retro perspective meeting at the end of the sprint, before initiating a new sprint (#2 Allen Holub). It should along with other specific feedback and suggestions of improvements, also be evaluated why the splitting was needed, as well as what impact this has on the order/size prioritization of stories (#2 Dina Dacquisto). As a result there will therefore be incomplete stories at the end of the sprints. Incomplete stories returned to the backlog, does not reward any *story points*[29] before they have met the DoD criteria. Incomplete stories should therefore be putted back in the backlog and then

[29] Story points were briefly mentioned as an activity of splitting tasks and allocating points to these subtasks in order to perform for example the calculation of velocity.

prioritized according to user value (#2 Gina Dacquisto). It is noteworthy that the PO's prioritization of the user-value might have changed, which should be accordingly reflected in the queuing of stories (#2 Bob Jacobs). There is in other words a continuous activity of queuing the relevant stories according to the newest formulated user-value defined. In relation to the Lean start-up methodology (Ries, 2008), it thereby reflects an inbuilt element to have continuous user review, and to move the continued development in a direction somewhat shaped by the user. There are however no direct findings in the groups whether these reviews are taking place at the very end of each sprint, or in contrary are considered a spontaneous ad hoc activity.

In congruence with a sprint that turns south, the team might also experience sprints where it is needed to add more work, it has to be assed in the retro perspective whether the sprint is too long, and/or why the complexity of the task was wrongful estimated (#7 Kevin Kriner). It should ideally be up to the team if they want to add more work next time or shorten their sprint (#7 John R. Durgin). It has thus to be realized that project estimates, which the splitting of stories and allocation of story points effectively is, should be expected to be somewhat inaccurate because they are based on ever-changing assumptions. In relation, the more ahead in time there is estimated, the more the accuracy will decrease (#7 Pete Morris). In this sense it might also be a solution to shorten the sprint length, as a sprint of for example 4 weeks will have a very slow feedback loop. In that sense shortening it to 2 weeks, and further prioritizing *nice to have* in contrary to *must have* might make it more convenient to estimate accurately (#7 Reeju Srivastava). Finally as already noted, splitting helps planning, which can also provide better grounds for performing estimations (#16 Christopher Bimson). To split stories should be connected to the acceptance criteria, describing the requirements the product or project must meet. It should also make sure the workload could be managed within a reasonable sprint time-box, which others also argue ideally should aim to have a feedback loop on 2 weeks (#16 Allen Holub). The point that Scrum is not a rigid management practice, thereby illustrates how a team might need different sprint lengths. This aspect, and the tension this opinion might have with the surrounding organization will be discussed further in chapter 5.

These aspects should ideally be up for discussion on the retro perspective meetings.
As a general rule, all persons that are involved in the subject of discussion in the retro perspective should be there. In effect this would often mean combinations of the team and PO and SM (#8 Marcel Blok). It is thus crucial the team has a high degree of ownership in the process and expresses an interest in involving also other relevant parties. This could be departments/functions within the organization, or other stakeholders outside the organization

such as customers or suppliers (#8 Jim Kingsberg). It is important to note, how the team might not feel free to speak their mind in a setting of management or other non-team parties, which then in essence undermines the process-ownership and in that way the hole idea and power of the retro perspective (#8 Stephan Kristiansen). As will be presented in the following section, the same acknowledgement goes for performance metrics, which needs to address elements, which makes sense for the team.

4.3.3: Performance metrics

The selection of measure should therefore very much depend on the team and the context, in which it is not a given necessity to have for example both story points and task estimation. In regards to the findings on practitioners thoughts on performance metrics, these are very similar to the previous mentioned findings by (Kupiainen et al., 2015). Even though defect counts or customer satisfaction as the only measures are not directly mentioned in the findings in the community, both of these seems very implicit on the remarks on quality control and persistent of user value analysed above.

Just like (Kupiainen et al., 2015) found, there seems to be a consensus on velocity and effort estimation as the most beneficial to use. One explanation of how the relationships between these two metrics are, states that based on the organizational frame, a team might initially start to use story points. If the team wants to, they can begin add tasks, and eventually also begin add estimates to those tasks (#4 Kevin Kriner). Task estimation is described as being used prior to the development, in order to negotiate budgets, schedules and risks. Velocity is only relevant after the project have been approved, in order to track the team progress. There should however be a continuous spillover-effect in which this tracking also improves the accuracy of the task estimation (#4 Donald Reifer). The task estimation measures can also be combined with Kanban in order to limit WIP and thereby reducing risk and waste within the sprint (#4 Vishal Chapekar).

In larger entities it can be recommended to use story points as velocity on the team level and rate of progress at the subsystem level. This measure of rate of progress was not mentioned directly in the study of (Kupiainen et al., 2015). Put short it is a way of rating the ability to estimate by assessing the actual progress in relation to the planned progress[30]. The comment that it is relevant for larger entities, seems to be a perception with strings to the Leagile logic of the progress of going from Agile on a team level, to integrating planning elements of Lean in subsystems (Wang et al., 2012). Following the same logic, the measure of earned value is

[30] For more info on this measure see www.scrumalliance.org

mentioned to be useful at a project comparison level if there is more than one development team (#4 Donald Reifer).

In terms of who decide what to measure, it is also pointed out that the SM should be the least qualified to make estimates (#13 Andy Wootton). The SM should instead facilitate the process in the best interest of having estimates in congruence with the interest of the team of *"Not working to someone else's estimate"* (#13 Sergio Seelochan). A SM unwilling to involve the team in this process might be a red flag warning sign for the Scrum process (#13 Derek Fields). In order to measure the accurate average velocity (measured in stories-per-week), the stories has to be split into smaller stories of preferably a few days. To break down epic stories into smaller rocks is in that sense facilitating a very frequent agile iteration of incremental delivery (#16 Paul Oldfield). As good as epic stories are for planning on higher levels, they are equal as bad for the straight development purposes, in which they have to be split (#18 Paul Eastabrook).

In line with the definitions of (Poppendieck & Poppendieck, 2003), waiting user stories is like a wasteful inventory, and should be visualized and reprioritized in order to minimize it (#18 Michael Kusters). On this visualization, which is very much similar to Kanban, the stories should be highlighted as blocked and waiting for input (#18 Paul Eastabrook). For this purpose to map the WIP is in line with the traditional throughput thinking (Al-Baik & Miller, 2014) recommended to highlight the levels of work in process as well as queue sizes (#18 Broughan Macklin). In the following section Kanban will be further elaborated.

4.4: Kanban

In this section the usage of the community thoughts on the usage of the Kanban system and furthermore it relation to Scrum and sprints will be analysed. It is no secret that also within this area the community confirmed how it uses a mixture of different ways of utilizing Kanban, in line with the 5 different categorizes stated by (Al-Baik & Miller, 2014).

4.4.1: How to use the Kanban System

Within a Kanban visualization system, the product backlog use a list of all the requirements which depending on context can be made by the PO, development team and/or other stakeholders (#5 Tushar Jain). Kanban can take many low-high tech forms ranging from blackboards, paper sheets to very advanced virtual performance and documentation oriented

software applications[31] (#5 Abdt Wiittib). The process of selecting elements from the backlog, prioritize them for development and test can take many different documentation forms. Writing code straight from requirements is appealing for XP programming, whereas combining software applications of Jira and Confluence makes reporting to stakeholders an automatic event (#5 JL Heather). In line with the findings of (Strode, Huff, Hope, & Link, 2012) in terms of co-location, the level of detail of requirements should very much depend on the team, for example the required documentation could depend on if the team is distributed or collocated (#5 Andrew Tomin). Whereas the documentation in the virtual software application is more extensive, it also presents great advantages for virtual-collocated developers and performance measurement (#5 JL Heather). There are also potentially enormous productivity gains derived by spillover-effects with other software tools (Andrew Tomin). These potential effects will be examined in-depth within the case-study findings in the next chapter.

To assess the relevant usage of Kanban & Sprints, some key indicators are: 1) incoming demand and 2) the current throughput, which should be looked at as *your capability to fulfill that demand* (#19 Raazi Konkader). In essence this context, characterized by the difference between the Service-Level-Agreement (SLA) and the Operational-Level-Agreement (OLA), should define to either use Kanban's iteration less flow vs. Scrum's time-boxed format demand (#19 Raazi Konkader). It is thus also an opportunity to have teams running in different cadences, in which Kanban can be used for the unpredictable demand and Scrum for the predictable. It should also be noted that using sprints according to Scrum is pointless if the team is not really looking into iterations as well as learning from the feedback during a retro-perspective. Kanban should be used for complicated work, whereas Scrum is for very complex work (#19 David Denham). If at least 50 % or preferably 80 % of the work for each sprint is unknown, there is next to no point in having a planning meeting in Scrum (#19 Paul Wintjes). In terms of strict incident management with very high inquiry volumes, it might not be ideal to run any agile methodology, and this support service should instead be handled by a more standardized operation approach (Sean D. Mack). On the topic of how to divide these responsibilities between the development and operations department, it is now time to elaborate a bit on the DevOps role in this regard.

[31] Among others counting Jira, TFS, Rally, Spreadsheets, Confluence, Version One, EA & Sparx.

4.5: Dividing tasks between Dev & Ops

As much as handling support-items on the development team can be a source of waste, parts of on-going customer feedback should also be considered within the continued feature delivery (#3 Cheryl Ash). In a small development unit, in which a team is responsible for both support requests and feature delivery, the activities should be grouped together, but separated in order to allocate an appropriate amount of resources according to the vision stated in the release map. Alternatively, the teams might split up in two groups, in order to process support request through Kanban, and feature delivery through Scrum (#3 Cheryl Ash).

No matter if this is managed through a unified or divided team, or run by an operational department, the information needs to be derived, communicated and prioritized by the development team (#3 Sioen Evans). Who the responsible for this should be depends on the setup, in which either the PO or an appointed DevOps associate should make this process run smooth. In some companies the software engineers are rotated on a weekly basis to have primary DevOps responsibilities. The developer then leaves the sprint while *on DevOps call*. The issues they find in this role then become new items for the backlog, unless they require an immediate fix (#10 Michael Coen). Such an arrangement might be a compromise, which makes sure to enforce a task and process end-to-end ownership described by (Floris et al., 2014), which might lead the way to come close to a state of continuous delivery and deployment within the organization. A collaborative and autonomous DevOps culture is also within the community claimed to be possible between two self-organizing teams, in which there exist a very high precedence of evolvement and distribution of mutual valuable information. At an early state it might also be valuable to have developers involved in support, as they in this way will very fast have a strong incentive to develop a sustainable solution that minimizes the need for related support inquiries. Later on up to the MVP state, the whole team should be involved in customer feedback (#3 David Cooper). It has in other words to be intentional how much of the sprint is allocated to feature delivery, integration and deployment management. These activities would thus in the case of being value-adding tasks be measured in the backlog. It is nevertheless beneficial to use the ratio spend on infrastructure maintenance vs. software development to asses if the work resources fit the prioritization vision (#10 Guy Maslen).

4.6: Conclusion

The aim of this chapter was to answer how software professionals in turn are combining Lean and Agile practices, tools, activities and performance metrics. First of all there was identified a consensus on certain aspects. The most outspoken of these were the need for a team to be provided with autonomy, craftsmanship and diversity in order to be creative and innovative (#1 Allen Holub, Brad Black, Chris Alexander). Furthermore the vast majority of the posts on the ideation process, stated it should be based on the definition of a user-case, which then would generate a discussion, leading to fast proto typing of the infrastructure and architecture needed (#6 Eric Tucker). More there existed a consensus on the difference of the fixed Scrum and continuous flow of Kanban (2# Allen Holub). To this end, it also seemed as a common understanding how Kanban is best suitable for the unpredictable demand and Scrum for the predictable (#19 Raazi Konkader, #19 Paul Wintjes). In regards to Scrum, there were a variety of opinions on how rigid vs. flexible a process it should be. This debate centered on the applicability of the need of using all or only some elements of planning, daily-standup, review and retro perspective. How stringent to follow the prescribed roles and these main pillars, was however also acknowledged to be a matter of the organizational dogma and maturity level of the team and management (#15 Paul Oldfield). It was thus very outspoken how the Scrum master should remove the impediments to flow, and not have a project manager role of focusing on deadlines (#12 Tom Mellor). This aspect was much related to the consensus on how the entire team preferably should have as much process ownership as possible, which should be enforced by an end-to-end evolvement (#9 Michael Farag, #3 David Cooper). How well this entire process should be documented, seems influenced a lot on whether the team is distributed or collocated (#5 Andrew Tomin). The level of documentation and planning, especially among teams, was by some practitioners associated with the addition of Lean to the Agile practice, close to the archetypes define by (Wang et al., 2012). In relation to the end-to-end evolvement, there was a very fragmented opinion on the role developers should have on especially the aspects of ideation, integration, deployment and customer support. Nevertheless the community agreed on the need to have the team involved in the decision of deciding how much of the development process was allocated to feature delivery, integration and deployment (#10 Guy Maslen). In relation to this DevOps element, this were very much project and organization dependent (#10 Michael Coen). This contextual aspects was also the case for the utilization of performance measurements, in which velocity and effort estimation was found to be most widespread, in line with the findings of (Kupiainen et al., 2015). Except a few comments on the synergy effect of tools, there were not offered in-depth explanations or examples.

CHAPTER 5: MULTIPLE CASE-STUDY

5.0: Overview

This chapter will start with a brief presentation of the four selected software development companies, which will be followed by an introduction and overview of the findings in each of the selected case companies. These findings will next be elaborated in a cross-comparison of how the different steps in the development process are approached. This analysis will also draw and discuss aspects in relation to findings from the previous chapters when appropriate.

5.1: Case selection and overview

In an attempt to present different approaches to software development, the case organizations depicted in figure 16 are all developing very different product and/or services for their clients.

Name	Type	Employees	Services	Interviewees
Vizgu	Start-up	6	Picture-recognition app	2
I3 Consult	SME	3	Development & maintenance consulting	1
Brainbot Technologies	SME	20	Blockchain development & consulting	1
Siemens Denmark	MNC	7.800	Extensive engineering & operational services	4

Figure 16: Selection of cases. Own Creation.

Their respective development processes are accordingly rather contrast-full in comparison, also influenced by the varying organizational conditions as well as shaped by their unique niche conditions. Within this sampling it thus has to be noted that Brainbot Technologies and I3 Consult are thus similar to the extend that they both receive their primary revenue flow in delivering services within entities in the financial sector. None of the selected cases are public

or non-governmental entities. Except a few notes on the compliance aspects within the development of software to public sector by developers within Siemens Denmark, the public sector is thus a missing perspective within the sampling[32]. The same can be said about scientific research and development conducted at universities. To this end Vizgu is collaborating with researchers from a poly-technical university located in France. Moreover Brainbot Technologies is in their development of open-source software collaborating with the Ethereum Blockchain Foundation as well as several high-profiled universities globally.

5.2: Presentation of the four cases

In this presentation of the cases the charts will only summarize the finding which will be analysed in the comparison of the cases.

5.2.1: Vizgu

Vizgu is the name of a Danish based start-up[33] which has been developing and testing its picture-recognition technology for about 2 years. The vision of the team is to aligning the bare print of the physical world, with all the online opportunities the Internet offers. The core concept of Vizgu is in that see to establish interconnectivity among visual objects encompassing a cross-platform application, with the ability to offer new user-cases based on picture-recognition. The basic idea is using picture-recognition as a means to make entire pictures into QR codes, in order to deliver predefined features to users scanning pictures on mobile devices (168). These features to date include the categories of social media redirecting, layers of music & video, contact features as well as including standard QR and letter recognition code compatibility. The range of these features can be extended into whatever other elements one could think of already exists on the Internet. The data on Vizgu is based on the experience from the two lead developers, which were collected in an interview on the 7/6/2016. Figure 34 in appendix 7.2.2 offers an overview of the findings at Vizgu.

In short, Vizgu is committed to a high iterative development process based on primarily agile principles (233). The development inputs are integrated and tested several times a day according to XP best practice. The tasks are defined as through design driven features, and split in a semi-systematic story format based on specific user-cases within a virtual Kanban

[32] It was not possible to grant access to in-house development within the public sector in Denmark. Some invitations to participate in the studies were rejected while the majority referred to their private IT partners such as KMD.

[33] Vizgu have been categorized as a start-up as it falls into the stereotype of a garage development process.

board (84). Although Vizgu is clearly a start-up, it is nevertheless not following the Lean Start-up methodology of creating a MVP in order to get user-feedback; instead the nature of the feedback can be characterized as a technology-push. The entity mentions they furthermore been using elements of Lean Start-up and the Business Model Canvas as a means to generate their business model draft.

5.2.2: I3 Consult

I3 Consult is a small Danish based consultancy, which within the last 3 months has worked on developing projects in the financial industry for among others Bank-Data Frederica (727), as well as currently working on 3 data-modelling and maintenance projects for Nykredit (724). The projects are very different in nature, ranging from small system changes with no stakeholders, into cross-organizational projects in which *"several systems have to be build as one, which means there are several stakeholders"* (719). Figure 35 in appendix 7.2.2 offers an overview of the findings at I3 Consult.

I3 Consult working processes normally takes place in 2 or 3 locations, with developers located in Kolding, Ålborg or Copenhagen (756). As I3 Consult thus typically have the majority of their workforce on-site, working shoulder-to-shoulder with their clients' in-house developers, the typical software development process is designed to fit into the methodology generally utilized by their clients. The interviewed CEO of I3 Consult states that they typically use *"a mixture of classic waterfall and agile inspired methodologies* " (728). When questioned further in detail, it is thus clear I3 Consult also as a rule of thumb uses other practices such as Scrum, and visualizing the backlog and development process on a virtual Kanban board.

5.2.3: Brainbot Technologies

Brainbot is a software development and consultancy serving primarily financial industry clients, who are in the process of testing and implementing blockchain based solutions (460). Brainbot has its headquarter office in Mainz, and another office in Berlin, Germany. There are additional virtual co-workers and sales representatives in Copenhagen, Florianopolis, Belarus and Mumbai (475). Figure 36 in appendix 7.2.2 offers an overview of the findings at Brainbot Technologies. The interviewed developer from Copenhagen has both development and sales representatives, as he is responsible for all of Scandinavia, and thereby *"Attends several of conferences and tries to get contracts or other different leads"* (519). In turn the CEO of Brainbot thus is setting a frame for the processes and price of estimation for a project, in which the developers can negotiate within (633). The interviewed developer

described the solutions, as "*they are commonly very unique, because it is often a customer who would wish to have certain functionalities, or a process in their corporation that they would like to have transferred to a blockchain*" (615). Brainbot thereby operates through the business model of developing open-source software, which they use as their building block (616), with the revenue stream of assisting/educating clients with the building of successful proof-of-concept implementations within their respective business areas (466). Brainbot don't have any suppliers, but are actively collaborating to integrate their technology with other open-source providers, in order to "*improve the opportunities of the ecosystem* (683)".

5.2.4: Siemens Denmark

The interviews conducted in Siemens headquarter in Ballerup, Denmark, are based on a specific development process, as well as taking a more general perspective on Siemens software development approach. For the specific development process a HR executive product owner as well as a project manager for 6 developers was interviewed based on a fixed 10 week development project of an internal HR on boarding website (1047). The site is currently being reviewed to be adapted on a global scale within the relevant Siemens business divisions. Figure 37 in appendix 7.2.2 offers an overview of the findings at Siemens Denmark. Furthermore, Siemens general software methodology was uncovered through the interview of two senior overlords (1214), who were interviewed about their experiences and opinions on combining various aspects of Lean and Agile on a range of very different global projects within the Siemens Conglomerate. They have, with a humoristic undertone, been giving the title of overlords by the developers, as they have also been coaching project mangers, scrum masters and product owners in Lean and Agile methodologies within Siemens.

5.3: Comparing case findings

In this section the end-to-end development process will be analysed on the basis of a selection of illustrative examples derived from each of the cases. For clarity the end-to-end development process has been divided into the four parts: 1) Finding & validating ideas, 2) Day-to-day development, 3) Testing, integration & deployment and lastly 4) Performance evaluation and process improvements.

5.3.1: Finding & validating ideas

The four examined development processes have demonstrated very different approaches to how an idea is identified. This is also true further in the ideation process, in which the cases

provides different perspectives on how ideas are accepted, and further on how the coordination of breaking the delimited idea into subtasks is taking place.

In the one end, the case of I3 Consults illustrates a clear-cut consultancy process, in which the client defines the overall tasks. The CEO notes that: *"the business set some demands, so we have some product owners out there, who says we need to see this, and this and this"* (741).

I3 Consults process is in that sense very much based on the Scrum role prescription, in which the product owner sets the overall demands, in some cases in a lose-formulated fashion or vision format. It is then up to the consultants in I3 to frame and serve the demand by drawing on resources within the client's organization. I3 Consult would then generally be assigned 4-5 in-house employees who are responsible for granting access to the specific required internal subsystem (747). The CEO describes this part of the process as:

"We then go and look at where we can get those things here, and then we contact all those sources, who sits in in different systems, who could be interesting for us, so the product owner sets the demand, and we are then the execution part of the project" (745).

I3 consultants are thereby assigned a coordinator role in other to better frame and oversee the delivering of the abstractive high-level demand triggered by the vision of the product owner. In that sense there are indicators that I3 Consultants thus take some form of a Scrum master role in the process, and in that way facilitates the process. It has to be noted that I3 in addition also themselves bring in highly specialized developer personnel, and thereby are contributing with both management and developer resources to the project. As will be disclosed in the following case-examples, this blurred management/developer role seems to be more the standard than the exception within the sample of cases. In connection the on boarding website in the Siemens case offers an example on how this idea generating and selection phase unfolds one step earlier, that is from the perspective of the product owner. As Siemens had on a group level since 2012 had the on boarding of employees as a HR priority, this had led to the collection of a lot of feedback from employees. These employees had explicit stated what they would have liked to have known in advance before starting their position (1052). The HR executive described her role at this stage as a product owner:

"I had a vision about what I wanted; to have build a site, which should be targeted at our employees. We had some ideas for content, but besides from that my wish was to be inspired, and I needed several proposals on how such a site could be build up" (883).

Thereby illustrating a high-abstractive vision, which was up to the project manger and the development team to interpret. The project manager and the product owner, would then be working with a team which they had not previously been working with, which led to a phase lead by waste-full confusion, because the development team did not have a clear idea if they should bring day-to-day questions to the product owner or the project manager with ad hoc questions (895). As soon as this was sorted out and clarified on meetings, the project tasks were divided due to the developer's different skills within video, front-end, back-end and graphical design in a self-organizing manner (906). So although the developers were not used to working within Scrum roles, which led to confusion on who were setting the development specifications, they could nevertheless still manage to self-organize in an agile manner as soon as this chain-of-command problem was sorted out. The problems the team encountered, thereby seems to be connected to the lack of a shared understanding of the communication hierarchy.

In Vizgu the process of selecting and splitting ideas is in the same fashion initiated by the CEO, who brings an idea of a new feature to the developers. This new feature might have been identified in collaboration with the marketing employees. As were the case in Siemens, it is then very much up to the developer's interpretation to specify the technical solution. This aspect is from the developers point of view explained by the fact that the CEO:

"Does not have the same technical capabilities as us, he only specifies the idea and then we describe the technical aspects, and then we discuss what can actually be done, and how we think it is a good idea to make it" (67).

As the CEO in this way is a gatekeeper between marketing and the development in the ideation process, this might also lead to sources of waste in the communication the CEO facilitates between the two parties. The task descriptions for the entire team are, normally by the CEO, grouped in the backlog of a Jira Kanban board. The developers state how:

"They are then indexed as being marketing tasks or development tasks. Within the development tasks, they are also divided according to being server or client based" (128).

The tasks/stories are not split, but is in that sense very broad in their description and acceptance and definition of done criteria. In terms of the task estimation, it is handled very informal, also characterized by the coding style of the developers, in which the task overview is done when all the features required for that story is defined within the code. Deadlines are

then negotiated with the rest of the teams based on the next meetings as well as the scope of time the developers can dedicate during that given period (232).

In Brainbot technologies the respective developers are responsible for getting in users and in that sense also to get in their future tasks themselves. In turn this is only true for one part of the developers, as they internally rotate between improving the open-source software or being out finding and educating the clients. The developers in contact with clients, are responsible for both finding clients and defining user cases, as well as to lead and educate a number of in-house developers, in order to redesign the client's processes based on Brainbot's open-source technology. For Brainbot there is thereby a big difference on how the ideas for the development of its own open-source software is handled, in contrast to how it unfolds in the development of solutions at the clients side. On the client's side, Brainbot approach seems very much similar to that of I3 consulting, in which they adapt to the internal practices of for example using release or time maps, by the clients that have hired them. A big difference might be that the blockchain solutions that Brainbot offers are so radical and complex that there might be a big element of asymmetric information in the process. It can be argued that Brainbot to a higher degree than I3 sets the agenda to the product owner in terms of the relevant user-cases for what is even possible with a technology such as blockchain.

At the in-house development of open-source software Brainbot splits issues in Github, in a fashion equal to that of the splitting epic stories into smaller stories (596). These are based on own ideas, which might have been defined in a collaborative fashion on the daily stand-up meeting (549). The interviewed developer from Brainbot has a facilitator role on these virtual held daily-stand ups.

5.3.2: Day-to-day development

Whenever the vision or tangible nature of the idea(s) has been transformed, delimited and split into subtasks, the development process itself begins to take place. In turn it is very different from case to case how long the iteration from idea-development-testing typically is. Some influential aspects found in the cases in this aspect, seems to be the nature of synergy-effects from repository extensions, as well as whether the idea is split and assigned into tasks in specific time-fixed sprints. In relation the utilization of a pull-based Kanban system also seems to have an influence on the number and frequency of iterations conducted. All these aspects are of course also affected by the location of the involved developers, which

definitely have a strong influence on the way the communication among these involved actors, take place.

The process of Brainbot illustrates very well how combining repository extensions from Github with a virtual communication platform such as Slack, can be very powerful in a virtual developer environment:

"We start in Slack, that sort of is our base, which has a lot of extensions integrated. We have hooked up Github, so every time anyone commits anything, or make pull-requests, we can all see there is send notifications about it in Slack" (500).

In that sense by using the developer's own word, Slack becomes the center of the development process, in which they get notified anytime there are made any changes to the code. As will be examined in the testing section, this notifying function is also pushing messages into Slack whenever the automatic testing of the new code edit have been made. Slack also functions as an easy virtual chat-program, in which the developers can join sub channels to collaborate without disturbing the rest of the team. A similar, but still less systematic way of utilizing Slack in combination with Github, was found in the development of the Siemens on boarding site. Here all the developers were co-located 2 days a week, and in the rest of the time communicating through Slack and utilizing Github to automatically sent tickets and build updates to Slack (932). The iterations were happening in a changing frequency, with more than a total number of 600 releases on Github at the end of the project (1121).

So whereas the Siemens project used some of the synergy of Github and Slack, they also in contrary to the process of Brainbot used a virtual Kanban tool called Soup (945) to keep track of the progress in the backlog of tasks. To track the developers' progress, the Brainbot team thus simply use the issues committed on Github as an indicator to see the missing tasks as well as a way to look into the quality of the work that has been conducted (585). One reason why Brainbot as the only one of the 4 case companies are doing it this way, might be due to the fact that they are all coders, and when they are only assigned a few employees at a time per project, they can easily manage to track the progress of the tasks this way. In the case of for example Siemens or Vizgu, several non-coding actors needs to track the progress, which might have been the reason for utilizing a graphical progress tool such as Jira or Soup Kanban board. One example of such an actor is the project manager in the Siemens case, who would have a double role in the process:

"I have both been the customer that the development team, if you consider them as a supply, has been the customer that they are reporting to" (859). "*I have also been directly involved in the development of the project, especially in the initial phase of which we had to set the direction, and then they team said: 'okay we got it', and from there my role has been something like 'what do you think about this, what should we do here' – in that sense their daily contact"* (866).

At the same time the project manager & the product owner had contact at least every second day (886). The entire project participants then meet up once a week, in which the product owner and project manager toke a *"customer position in which we let them present their work and let them explain it"* (910).

In relation to briefing meetings, I3 Consult as the only one of the case companies uses the Microsoft's Link tool, to virtually share screen with each other. Link is used on I3's daily-stand up meetings, in which internal feedback among the participants is provided (833). Link also provide the feature of

"Letting the individual to go in and moving items on the screen, so it is not everyone that have their own screen, but it is locked, so they can only look at the Jira Kanban Board, and the participants can within these 10 minutes not look at emails or anything else" (755).

The Link software is in this way used as a control mechanism by the person who is presenting, in a fashion so no of the people attending the meeting can open any other windows than the Jira Kanban board. This example might also illustrate how I3 Consult is very plan driven, as they normally run sprints for 3 weeks, in which the product-owner specifies all the tasks that has to be developed, which through the formulation of a user story, contains all relevant subtasks (787). Back to how tasks are handled in I3 consult, they do not use the term epic stories, but looks at the user story, and then *"split it into atoms, and look at what the usercase include, and then we prioritize the order"* (793). I3 Consult is then typically having a work-size of each subtask between 1-2 hours, so they can easier be compared and tracked in Jira (796).

On the subject of balancing the length of the sprint with the nature of the process and organization, the overlord B from Siemens explains how it for him personally have been a tough battle to cope with the fact that Siemens on international projects for some reason had decided all scrum sprints should be time synchronized and all having the length of 3 weeks (1235). From his experiences this was the biggest bottleneck of having a scrum setup that fitted his process, because the team would have liked to have sprints varying from 1 to 2

weeks (1241). To this point the overlord B also brings in some of his experiences about experimenting in order to find the right sprint time for the team. He note *"So I think you should play with it, and see what works for you"* (1254). These statements do in other words describe the organizational planning trade-off, which sets some limits to the self-organizing aspects of utilizing the full Scrum methodology. In reality such organizational restrictions, which are often there because of compliance procedures, requires Siemens to run waterfall combined with Scrum. They have in that sense a number of defined iterations; all the source-code of these new builds are then locked until a number of tests take place (1301). In another example of an agile, non-scrum project the overlord A is responsible for he explains how it is now running through a Kanban process because it has few people involved, whereas he normally utilizes the main body of the Scrum methodology on bigger projects (1346).

Finally at Vizgu the two in-house developers are often working on different tasks, they more of less have the capabilities to complete the others work as well. As the developers are sharing an apartment, and have a very similar coding-style, they are extremely flexible to switch tasks, and often they both contribute to solving a story.When only collaborating the two, they therefore are very often co-located, in contrary to the rest of the team. In contrary to the other cases, Vizgu is not using Github, but instead Team explorer's visual studio as their repository to store and test the code (176). According to the developers this gives the advantage of *"making some code, in which we can 'roadbacke' if there is something that should not be there, or be if something went wrong. We can thereby see, how often people update"* (181). As long as the bug can be reproduced within these tools, it is then quiet easy to isolate and correct it. Furthermore the team uses Dropbox to distribute the codes achieves, in turn to distribute the alpha product with the rest of the organization when the development cycle is completed. In Vizgu this distribution does however not equal a final acceptance of the code, as the *"most things are still very open after we have finished a coding cycle"*. In Jira this achievement is illustrated by removing the task from the box of *in development* into *ready for testing* (276). If the testing is then completed without any notes for editing, the task is finally removed into the category of done (279).

5.3.3: Testing, integration and deployment

The cases demonstrate a variety of testing, ranging from non-coding based monkey testing, customized scripts and semi and full automatized build-bots. Furthermore user and peer-reviews are also utilized within Siemens and Brainbot.

In Vizgu testing is managed without a general system on an internal ad hoc basis (170). This involves testing of changes of minor tasks on Jira by the rest of the team or the developers. Among intermediate external stakeholders (109), this might also involve collaboration with the patent provider to the picture recognition algorithm Vizgu is leasing. This is determined by Vizgu has not released its beta, and therefore not having any direct users for testing and giving feedback (116). This aspect is frustrating for the developers, who state that:

"We sort of have 2 groups of users, in one the one is the one that is paying for a service, and in the other end freemium users that needs to have a service delivered for free, and we frankly don't know if either of the groups are interested in playing their part" (432).

The developers are in that sense highlighting the absence of user-feedback on Vizgu's minimal viable product, and further underline the uncertainty of customer validation in related to the remark: *"we actually have no idea it is going to spread like rings in the water, or if this another one of those, that does nothing*" (432).

In I3 Consult, they have testing templates, which are often customized in order to perform automatic testing. Namely if many developers have been working on the same code, it is then necessary to design test scripts to look into input and output parameters, and generally test if the things connected are taking appropriated actions and functioning together (826). In relation to the priority of the testing, the developer answer:

"We speak of a definition of done, and we are typically operating with 80/20. We want to reach 80 procent of the tasks that are submitted within the sprint, and the last 20 % is tasks that are 'nice to have'. The 80 % of the tasks that in contrary fall in the category of "need to have", are thus considered done when the pass testing in 3 different sequences, ranging from test and media to system environments". (812).

Finally, if the testers from the business areas are positive, I3 runs a verification test, which is the last test before it goes into the production, an area which I3 is not involved in any testing (816). This is however not the case with the overlords from Siemens, which both have a lot of experience in managing development and testing of project going from development into production. On this DevOps subject, the agile overlord B notes that:

"We had decided that we would not use a DevOps, because we wanted everybody on the team to be able to perform maintenance on the test-systems, and to keep track on our build-mechanisms as well as to manage Jenkins. This is still the vision, we however encountered the

test setup had been so complex so we need someone to coordinate the tasks, which then lead to the definition of this DevOps role" (1382).

The Agile overlord B describes a similar experience, in which they had to deploy the tested code to 117 different hardware units, of which they had to test them all with a semi-autonomous test script. In this setup a DevOps role was also needed in terms of documenting the test pipeline as well as making sure the testing-scripts were operating correctly (1371). The complexity of the test-setup thus in some cases demands a person to specialize in the area, who would document and delegate tasks to the other team members. Both of the overlords do however underline such a DevOps role is rarely needed on less complex projects. In relation to DevOps ambitions of systematic and continuous deployment, the overlords also go in depth with the difference of testing according to following continuous integration or continuous delivery. The overlord A explains how the concepts are very different even though they sound alike. He states that while you as a developer in continuous integration hand over a tested piece of code, the task is passed over to another person, and thereby out of your hands. Following continuous delivery, the source developer in contrary still has a full stake in the code cycle until it is done. The agile overlord B further exemplify his point by stating that

"In continuous integration you would get a message saying the testing failed, whereas in continuous delivery you would get the code back if it was not good enough" (1390).

From a process-perspective this does a big difference, due to the need having a reliable continuous delivery practice sets to process improvements. The developers responsible for the success of the code brings of optimization related questions of *"how can we do this better and faster"* (1420). This personal stake is present within continuous integration, in which the responsible is typically more fragmented, at it is not the developer who handed over the code, but another employee in another department who might have his focus elsewhere (1427).

A consistent point in this regard is to decide whether you would like to run continuous integration or delivery right away, because if for example a continuous delivery approach has been selected this should be build into the architecture of the source in the beginning. The overlord A explains how *"You only have to separate it all into components, and then think about how every component can be tested separately"* (1311). The testing procedures chosen for a given build, thereby states some basic rules for how the source-code should be designed in order to fulfil the testing criteria most efficient. In Brainbot the developer describes the testing-process as:

"We have a program that is called Travis, which is a build-bot, so every time something is pushed into our Github, it goes through the entire build and send a notification to Slack whether the build has passed or not" (504).

It is in that sense a continuous integration tool, so every time anything is committed to change the code, it goes through all the new code and run all the tests (509). Furthermore as Brainbot is active in the blockchain open-source ecosystem, this also presents some testing opportunities in terms of benefiting from external peer-reviews:

"One of the systems we build in is Python Client of Ethereum, and there we experience a lot of peer-review from many external parties who submits pull-requests, with their improvements to the code, in some of the newer systems there are not yet so many peer-reviews, because there are not many people who knows and uses the software, but that will properly come (711)".

At Siemens, a project normally also goes through 3 test environments, in which the developer can normally get near to instant feedback from his unit testing, compiling and integration testing. Depending on the security and compliance level of the project, it then might go through a complex release test setup, which might need 3 days to provide feedback (1411). On the on boarding project in Siemens, peer-testing was not really aimed at looking into the quality of the code done on the website, but rather a way of getting relevant feedback from the future users of the site in the business areas:

"*Quiet late in the process we asked especially within HR, but also in the business division, that is the ones who produce wind power and building different technologies. We asked every business, 'we are about to make an introduction about you, please give us 5 sentences about the most important things about you'.*

In that sense every business area had a chance to provide input on how they should be introduced (1067).
The project manager notes that:

"It was a really educative process, which we perhaps should have initiated earlier on, because it took some time to adopt and say, okay now the structure is confirmed, we are now able to finish the layers of the storyline" (1082).

The feedback process was designed to inquire feedback from a mixture of different employees within the business areas, which reflected an opinion quiet different than from the feedback generated within feedback from the HR department. The product owner elaborates: *"It is the business areas there are supposed to be using the sites, so it is crucial they believe it is valuable for their new employees to use"* (1090). The user-feedback derived from the business areas did in that sense overrule some of the insights provided by peers within the HR department in order to create an output that the business areas would be more likely to have a stake in using.

The overlord B from Siemens also elaborates how he often host prototype reviews for the clients of the different projects, for example the S-Train Company in Copenhagen. Despite these users are typical executives, and not hands-on employees, both developers note that they are very good to participate and ask relevant questions (1450). When asked further to the background of the participants, they would ideally like to have both the technical hands-on employees who can understand the build, as well as the executives who have the decision-making competence present at these reviews.

5.3.4: Performance evaluation and process improvements

I3 Consult conducts retro perspective after each ended sprint. The retro perspective meeting normally includes all the project participants, which would normally be the product owners, in-house system responsible employees and 2 consultants for I3. (844). Although the Jira Kanban delivers performance measures on velocity and burn down charts, this data is rarely evaluated, also due to the fact that the tasks on Jira are not split systematic. Instead the CEO from I3 notes:

"We get measured on if we manage to live up to having all the tasks done within the assigned sprint, that is an indicator the managers like" (805). This statement furthermore refer to the 80/20 principle for the sprints, in which that target is to complete 80 % of the tasks within each sprint. The performance evaluation at I3 in that sense has its focal point if the development team has been able to live up with the targets that were agreed on in collaboration with the product owners on the initial planning meeting. As there are pending dialogue throughout the process if any errors come up, both the expectations of the product owner and I3 can be adjusted before the retro perspective. The evaluation whether a project have been a success or not, is in than sense a quite qualitative judgment, based on the shared responsibility on the accuracy of the initial planning meeting. The CEO from I3 further states that their budget targets are handed over to another department in fx Nykredit, who evaluates

the targets allocated for the consultants. How this budget target analysis in turn takes place is however not something that he is aware of.

As Vizgu are not committing to Scrum, this also goes for process improvements, which are discussed some time when all the team is united for working in a beach house in weekend, or by the CEO who introduces new tools, such as the Kanban board. In Vizgu the performance measures mentioned above which are auto-calculated within the Jira Kanban software are not utilized in any systematic manner. The Jira Kanban's function in this relation is thus reduced to visualize what all the parties of the start-up are working on. So even though none of available traditional quantitative performance measures are utilized, the Jira Kanban board is still indented to visualize how many tasks each team members have completed, how many they are working on now and for how long they have been doing this. Furthermore both marketing and the development team upload the finished proof-of-work to Jira, which in that sense also serves as a Dropbox alike knowledge library in which you can see all work completed. The developers both state they like this transparency (200). It can thus be hypothesized whether this utilization of Jira have an effect of the productivity within the team, such as regular performance measures intends to do.

When asked about performance measurements, the developer from Brainbot notes *"There are not a lot of that if I should be honest, we don't dig so much into that, perhaps my boss do, but it is anyway not anything that he goes openly about, it is a very relaxed attitude to have to those things"* (601).

From his own personal view, he further elaborates how this casual attitude determinates how he performs, because he do not like a high-performance environment based on fear of falling short from targets. Instead he notes that his development motivation *"is not based on that you are feel obligated to do it, but simply because you enjoy the people and what you are working on"* (604).

Furthermore Brainbot has status meetings *"every time, a week or something like that, on the projects we are working on"* (579). Depending on the workload, these status and improvements meetings can be less frequent (665). Even though all from the team can contribute with inputs to the others feedback, they are usually so specified that: *"Normally I don't stand and walk about a task which I don't work on"* (543). As long as the developer team is as small as 6 persons, it is thus still considered valid to have all to participate on the meetings even though they work on different tasks (546). At Brainbot's workcation trips to

places such as Hongkong or Shanghai, they work a lot, but they also socialize and go out a lot (671).

On the differences of Lean and Agile in relation to process improvements the overlord A from Siemens notes: *"I actually think most of it has strings to Lean"* (1430). The developer B elaborates this point with his impression that a Scrum retro perspective is designed to improve the Scrum process, which goes very well hand in hand with a broader Lean outlook on how to improve the surrounding process. From his point of view the elements are actually so interrelated that *"We often don't know we are doing the one or the other",* after a small break he concludes *"And therefore I think it is a little bit silly to make a difference on the two concepts"* (1439). The basic point here in this regard is in when the involved retro perspective participants are having a meeting, and the discussion moves into a debate whether to shift the build and testing into continuous delivery instead of continuous integration, it is according to overlord B *"It would then be to adapt extra Lean elements into the process, but that they don't care about that"* (1441).

So according to the overlords from Siemens, the developers are to their experience not concerned with how things are labelled, rather than what actions these labels ultimately leads to. It is thus very crucial to set clear expectations, what the intended actions of labels put on organizations or projects are. This point is illustrated with some of the hurdles they've encountered in Siemens themselves, where

"When you come to a developer and say that you are having an agile project, then they expect that you have daily-stand meetings, and besides some meeting every 2 or 3 week I can sit and code means you can code and then have meetings, when this is not the case they become very frustrated due to their understand of the agile label" (1317).

So as much as there are no clear distinctions among some of the more high abstractive elements of Lean and Agile, it is thus clear what expectations developers have to the level of self-organizing associated with Agile. It is thereby underlining how the conversation should not be about the high-level abstractions of the concepts, because different employees will most likely have different perceptions of what they encompass. Instead to set a clear direction for the outlay of the development process will make sure to have a common understanding of how the development process will take place.

5.4: Conclusion

The objective of this chapter was to answer how and why software teams are applying Lean and Agile in different situational contexts. First of all the findings of principles, practices, activities and performance metrics in the four cases, demonstrated development processes based on four different ways of combining the concepts. It was very outspoken how all cases started the ideation process with the definition of a vision or tangible usercase, which where then in all of the cases very much up to the technical superior developers interpretation to provide a possible solutions for. With the exception of Vizgu, this process was initiated in all the cases with a direct user-need. The novelty of Brainbot's technical capabilities, illustrated in comparison to I3, how this encompassed Brainbot to a higher degree setting the agenda for the framing and development of the client's usercase. Without explicit mentioning Feature-driven-development, all cases would in turn break the usercase into features. The structured nature of splitting these features into smaller stories/subtasks was very different from case to case. In the structured end, I3 had a clear policy of splitting stories *into atoms* of a general length of 2-3 hours. This was explained be due to the estimation accuracy, which also thus was discovered to be the performance measurement they indicated their clients were the most concerned with. The same seemed to be the case in the Siemens on boarding project and Vizgu, nevertheless in a less structured and self-organizing manner. Finally in Brainbot this was very much up to the developers themselves, and would normally be something happening through the coding. As there were only coders working in Brainbot, this would also go for performance measurement, which in essence would only be the proof-of-work submitted to Github. All the cases demonstrated synergy-effect among software tools, in particular in relation to visualization, communication, knowledge-libraries and testing. Without going in depth with all these combinations, Brainbot's usage of Slack, as an integrated platform was in particular noticeable. The same goes for I3 Consult's and Siemens structured usage of Kanban software such as Jira and Soup, which also auto-calculated performance metrics. How these performance metric do not seem to be fully utilized in any of the cases is thus another notable point. In relation to build-bots such as Travis and Jenkins, they demonstrated throughout all the cases the power of automatic testing. Whereas I3 and Siemens also described how semi- and customized testing took place throughout three different environments, and in relation how this on very complex project might require a DevOps individual or team. Finally in terms of performance evaluation and improvement, this was found the most structured in I3 and Siemens through their commitment to Scrum. In both Brainbot and Vizgu, this was committed on an ad hoc basis, which would often take case during workcation trips with both work and social related activities embedded. Finally it is

according to the Siemens overlords not suitable to differentiate between the concepts, but instead of these labels all should talk openly and align about the expectations for the process.

CHAPTER 6: THE FINAL CONCLUSION AND ITS IMPLICATIONS

6.1: Main Conclusion

Through a distinction between principles, practices, tools, activities and performance metrics, this thesis have contributed to the fields of Lean and Agile within software development in three ways. First of all sub-questions A and B documented the evolvement, current state and identified gaps in the literature (Contribution I). Secondly the empirical study of sub-question C provided insights into how practitioners understand and claim to utilize Lean and Agile on an international scale, and discussed how this was in congruence with the findings in the literature (Contribution II). Thirdly the investigation of sub-question D evaluated how and for what reason a variety of elements of Lean and Agile co-existed in four different development processes (Contribution III).

Contribution I found that Lean and Agile within software development, individually and combined, were commonly categorized as being in a nascent state (Al-Baik & Miller, 2014; Wang et al., 2012). While the study found this to be true, especially in regard to the specification of process elements such as tools and activities, it also showed how hybrid data was increasingly used by academics in an intermediary and mature research fashion. As a result of the historical integration and progression of the practices, a high degree of similarity was found in the guiding principles of Lean and Agile. While this was the case, the literature review at the same time demonstrated the existence of a range of different opinions on the categorization and relation among the concepts (Ajay Reddy, 2016; Peterson, 2010; Poppendieck & Poppendieck, 2010; Wang et al., 2012). In the synthesis of the findings of the principles and practices, this study further hypothesized how the rise of DevOps might be a way of s*eeing the whole* on Agile teams, and in that way establishing a traditional Lean oriented end-to-end process flow.

Contribution II supported (Al-Baik & Miller, 2014; Rodríguez et al., 2012), by finding Scrum and Kanban to be the most popular practices within the examined LinkedIn groups. Although there were close to no explicit search findings on XP and Feature-driven development, the content analysis found how the elements of continuous integration and splitting of usercase according to a feature driven architecture embedded in a large amount of the posts. Suggesting how these practices to some extend might have become sub-activities in practitioners' conduct of Scrum and Kanban. The content analysis also found how Kanban was commonly understood to be best suitable for a complicated unpredictable demand and

Scrum for a complex predictable demand. In general the practitioner's perceptions of Lean and Agile as well as DevOps were fragmented, and a main point was to pick and choose from the concepts depending on a range of contingencies.

Contribution III demonstrated how and why these contingencies were shaping the development process within the four selected case companies. All cases were found to start the ideation process by defining a vision or tangible usercase, which with the exception of Vizgu, was initiated by a direct feature driven user-need. All cases also demonstrated how the difference in business and technical oriented capabilities among project stakeholders, often had a high impact on the step of preparing an idea to development. The cases also illustrated how this distinction also seemed to influence the nature of the usage of performance metrics. In addition to this the organizational lifecycle and culture, also seemed very influential on how performance improvements where handled through a structured formal process or taking place in informal workstations. The cases also demonstrated a range of synergy-effects by the utilization of testing, communication, measurement, repository and visualization tools. Finally the overlords from Siemens both underlined how setting clear expectations for how the development activities unfolds, are much more important than discussing Lean or Agile labels.

6.2: Managerial implications

In order to achieve a human-centric and craftsman-based state of flow within development teams, several aspects have to be taken into account. The software industry is, from my point of view, a pioneering niche within the discipline of new product development. It illustrates how many tasks in the future will be automatized, and at the same time how yet others become even more knowledge and innovation extensive. To understand the benefits of having the right amount of self-organizing planning and process ownership, evaluation and improvement is in that sense something practitioners in all industries can benefit from. Instead of engaging development processes based on a command-and-control logic, management should in line with the facilitator role of a Scrum Master, considering how it can remove impediments to flow for the team. Instead of guessing what this might be, the answer should be derived from a conversation on expectations, in order to find the right methodological fit for the expected outcome the team is supposed to deliver. The selection of the process layout, tools and metrics should in that sense be a shared negotiated decision of which all participants thereby can be held accountable for. As much as it is a good idea to try out new practices, there is a limit to the instability a team can cope with at a time. For that

matter it is crucial to pick-and-choose the elements of a practice that fits the status quo, while at the same time acknowledge there exists dynamics such as a planning meetings without a retro perspective in Scrum makes no sense.

Besides the maturity and experience of the team, as well as any taking organizational bottlenecks into account, an initial process conversation might center on the following 1) the size, stakeholders, complexity and length of the project, 2) Whether tasks is known and can be splitted 3) if the team will be working co-located 3) if daily-stand ups or visualization boards make sense 4) what level of testing is needed 5) if any collaboration with operation/production preferable? 6) do we have the right synergy-effect of our software tools?

When the team has agreed on these aspects, performance metrics should be utilized in order to give progression indications as well as to enforce cross-collaboration and process ownership.

6.3: Recommendations for future studies

With the strong diffusion of Lean and Agile elements among practitioners, documented in this as well as previous studies, there are rich and accessible data available to study a range of problems more in depth[34]. I would suggest to study the field in order to de-categorize the conceptual barriers among the concepts. I believe the low-practical approach of zooming further into the dynamics of tools, activites and performance measurements, is a beneficial way of crafting relatable and useful outputs for practitioners. As the newest research within the field are approaching the field through the utilizing of hybrid qualitative and quantitative methodologies, it seems evident the field has through the last 8 years moved beyond the nascent state as proposed by Dybå & Dingsøyr in 2008. Due to this evolvement, this might call for further integration of previously separate bodies of knowledge, as well as the formulation of focused research question in order to better document the performance outcome of a transition into for example Scrum or Kanban. Such data might thereby start to loosen up the clash of perceptions observed among fanatic agile developers and project management oriented business leaders.

[34] I believe self-organizing forums such as LinkedIn presents one of such opportunities of data to leverage further. Other alternative but rich sources might be meet-ups, conferences or other physical events which tracks alot of very enthusiatic and commited practitioners.

For establishing a strong appeal within both of these groups, such studies should from my perspective preferably be conducted in a longitudinal single or multilpe-case study format. These would ideally have identified focused quantitative measures, that would be collected for the case before, meanwhile and after a transition into a Lean and Agile inspired dynamics has been taking place. Furthermore, this transition process should preferably be studied through a qualitative lens in order to provide possible well-argued explanations behind the drivers of the identified performance outcome.

Finally I would like to encourage both practitioners and academics to look into integrating other academic fields to improve the rate of success for future development projects. In this regard I would in particular look into spill-over effects of integrating knowledge derived within neuroscience-based management, as well as how gamification might enhance aspects of self-organizing. In addition, the well-developed subject of how different individual motivational characteristics should influence the development team composition, as it also seems like a very interconnected field to integrate within Lean and Agile development.

CHAPTER 7: APPENDIX

7.1: Bibliography

Ajay Reddy. (2016). *The Scrumban (R)Evolution* (0th ed.). Pearson Education, Inc.

Al-Baik, O., & Miller, J. (2014). The kanban approach, between agility and leanness: a systematic review. *Empirical Software Engineering*, *20*(6), 1861–1897. http://doi.org/10.1007/s10664-014-9340-x

Brhel, M., Meth, H., Maedche, A., & Werder, K. (2015). Exploring principles of user-centered agile software development: A literature review. *Information and Software Technology*, *61*, 163–181. http://doi.org/10.1016/j.infsof.2015.01.004

Cantijoch, M. (2014). *Analyzing Social Media Data and Web Networks* (Vol. 26). http://doi.org/10.1057/9781137276773

Cockburn, A. (2002). *Agile Software Development*. Retrieved from https://books.google.dk/books/about/Agile_Software_Development.html?id=JxYQ1Zb61zkC&pgis=1

Cooper, R. G., & Dreher, A. (n.d.). Stage-Gate International and Product Development Institute VOICE-OF-CUSTOMER METHODS: WHAT IS THE BEST SOURCE OF NEW-PRODUCT IDEAS? Blockbuster Product Shortage. Retrieved from www.stage-gate.com

Danovaro, E., Janes, A., & Succi, G. (2008). Jidoka in software development. *Companion to the 23rd ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications - OOPSLA Companion '08*, 827. http://doi.org/10.1145/1449814.1449874

David J. Anderson. (2010). *Kanban - Succesful Evolutionary Change For Your Technology Business* (0th ed.). Sequim,Washington.

Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, *85*(6), 1213–1221. http://doi.org/10.1016/j.jss.2012.02.033

Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, *50*(9–10), 833–859. http://doi.org/10.1016/j.infsof.2008.01.006

Dybå, T., Sjøberg, D. I. K. D., & Cruzes, D. S. D. (2012). What Works for Whom , Where , When , and Why ? On the Role of Context in Empirical Software Engineering. *... on Empirical Software Engineering ...*, (7465), 19–28. http://doi.org/10.1145/2372251.2372256

Dyer, J., Furr, N., & Lefrandt, C. (2014). The Industries Plagued by the Most Uncertainty.

Harvard Business Review, *September*. Retrieved from https://hbr.org/2014/09/the-industries-plagued-by-the-most-uncertainty

Edmondson, A. C., & McManus, S. E. (2007). Methological fit in management field research. *Academy of Management Review*, *32*(4), 1155–1179. http://doi.org/10.5465/AMR.2007.26586086

Floris, E., Amrit, C., & Daneva, M. (2014). DevOps Litterature Review. In A. Jedlitschka, P. Kuvaja, M. Kuhrmann, T. Männistö, J. Münch, & M. Raatikainen (Eds.), *Product-Focused Software Process Improvement* (Vol. 8892). Cham: Springer International Publishing. http://doi.org/10.1007/978-3-319-13835-0

Highsmith, J. (2002). Agile software development ecosystems. Retrieved from http://dl.acm.org/citation.cfm?id=513727

Hines, P., Holweg, M., Rich, N., Browaeys, M.-J., & Fisser, S. (2012). The Learning Organization Lean and agile: an epistemological reflection. *The Learning Organization Iss The Learning Organization International Journal of Operations & Production Management*, *19*(10), 207–218. http://doi.org/10.1108/09696471211219903

Hoppmann, J., Rebentisch, E., Dombrowski, U., & Zahn, T. (2011). A Framework for Organizing Lean Product Development. *Engineering Management Journal*, *23*(1), 3–16. http://doi.org/10.1080/10429247.2011.11431883

Inayat, I., Salim, S. S., Marczak, S., Daneva, M., & Shamshirband, S. (2015). A systematic literature review on agile requirements engineering practices and challenges. *Computers in Human Behavior*, *51*, 915–929. http://doi.org/10.1016/j.chb.2014.10.046

Javdani, T., Zulzalil, H., Ghani, A. A. A., Sultan, A. B. M., & Parizi, R. M. (2013). On the Current Measurement Practices in Agile Software Development, 7. Software Engineering. Retrieved from http://arxiv.org/abs/1301.5964

Karlsson, C., & Åhlström, P. (1996). Assessing changes towards lean production. *International Journal of Operations & Production Management*, *16*(2), 24–41. http://doi.org/10.1108/01443579610109820

Kitchenham, B., Pretorius, R., Budgen, D., Pearl Brereton, O., Turner, M., Niazi, M., & Linkman, S. (2010). Systematic literature reviews in software engineering – A tertiary study. *Information and Software Technology*, *52*(8), 792–805. http://doi.org/10.1016/j.infsof.2010.03.006

Kobus, J., & Westner, M. (2015). Lean Management of IT Organizations : A Literature Review. *PACIS 2015 Proceedings. Paper 172*, 17.

Kupiainen, E., Mäntylä, M. V., & Itkonen, J. (2015). Using metrics in Agile and Lean Software Development – A systematic literature review of industrial studies. *Information and Software Technology*, *62*, 143–163. http://doi.org/10.1016/j.infsof.2015.02.005

Kvale, S., & Brinkmann, S. (2008). *Interview: introduktion til et håndværk*. Retrieved from https://books.google.dk/books/about/Interview.html?hl=da&id=wyB1PgAACAAJ&pgis=1

León, H. C. M., & Farris, J. A. (2011). Lean Product Development Research: Current State and Future Directions. *Engineering Management Journal*, *23*(1), 29–51. http://doi.org/10.1080/10429247.2011.11431885

Modig, N., & Ahlstrom, P. (2012). This is Lean: Resolving the Efficiency Paradox. Retrieved April 14, 2016, from http://www.amazon.com/This-Lean-Resolving-Efficiency-Paradox/dp/919803930X

Moe, N. B., Aurum, A., & Dybå, T. (2012). Challenges of shared decision-making: A multiple case study of agile software development. *Information and Software Technology*, *54*(8), 853–865. http://doi.org/10.1016/j.infsof.2011.11.006

Morgan, J., & Liker, J. (2006). The Toyota Product Development System: Integrating People, Process And Technology.

Mueller, R. M., & Thoring, K. (2012). Design Thinking Vs Lean Startup: A Comparison of Two Userdriven Innovation Strategies. *Proceedings of 2012 International Design Management Research Conference*, (January 2016), 151–161. http://doi.org/10.13140/2.1.1834.4647

Nicoletti, B. (2015). Optimizing Innovation with the Lean and Digitize Innovation Process. *Technology Innovation Management Review*, (March), 29–38.

Nonaka, I., & Takeuchi, H. (1986). The new new product development game.pdf. *Harvard Business Review*.

Pernstål, J., Feldt, R., & Gorschek, T. (2013). The lean gap: A review of lean approaches to large-scale software systems development. *Journal of Systems and Software*, *86*(11), 2797–2821. http://doi.org/10.1016/j.jss.2013.06.035

Petersen, K., Vakkalanka, S., & Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, *64*(C), 1–18. http://doi.org/10.1016/j.infsof.2015.03.007

Peterson, K. (2010). *Implementing Lean and Agile Software Development in Industry*.

Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., & Still, J. (2008). The impact of agile practices on communication in software development. *Empirical Software Engineering*, *13*(3), 303–337. http://doi.org/10.1007/s10664-008-9065-9

Pillai, A. K. R., Pundir, A. K., & Ganapathy, L. (2012). Implementing Integrated Lean Six Sigma for Software Development: A Flexibility Framework for Managing the Continuity: Change Dichotomy. *Global Journal of Flexible Systems Management*, *13*(2), 107–116. http://doi.org/10.1007/s40171-012-0009-2

Poppendieck, M., & Poppendieck, T. (2003). Lean Software Development: An Agile Toolkit.

Retrieved April 14, 2016, from http://www.amazon.com/Lean-Software-Development-Agile-Toolkit/dp/0321150783

Poppendieck, M., & Poppendieck, T. (2010). Leading Lean Software Development: Results Are not the Point. Retrieved April 14, 2016, from http://www.amazon.com/Leading-Lean-Software-Development-Results/dp/0321620704

Power, K., & Conboy, K. (2014). Impediments to flow: rethinking the lean concept of "waste" in modern software development. *Agile Processes in Software Engineering and Extreme Programming 15th International Conference, XP 2014 Rome, Italy, May 26-30, 2014 Proceedings*, 203–217. Retrieved from http://dl.acm.org/citation.cfm?id=2813544.2813558

Qualitative Data Analysis: A Methods Sourcebook: Third Edition: Matthew B. Miles: 9781452257877. (n.d.). Retrieved June 10, 2015, from http://www.sagepub.com/books/Book239534/toc

Ries, E. (2008). The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses: Retrieved April 13, 2016, from http://www.amazon.com/The-Lean-Startup-Entrepreneurs-Continuous/dp/0307887898

Rizvi, B., Bagheri, E., & Gasevic, D. (2015). A systematic review of distributed Agile software engineering. *Journal of Software: Evolution and Process*, *27*(10), 723–762. http://doi.org/10.1002/smr.1718

Rodríguez, P., Markkula, J., Oivo, M., & Turula, K. (2012). Survey on agile and lean usage in finnish software industry. *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '12*, 139. http://doi.org/10.1145/2372251.2372275

Salah, D., Paige, R. F., & Cairns, P. (2014). A Systematic Literature Review for Agile Development Processes and User Centred Design Integration. *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE '14), London, United Kingdom, 13-14 May, 2014*, 5:1--5:10. http://doi.org/10.1145/2601248.2601276

Schwartz, H. A., & Ungar, L. H. (2015). Data-Driven Content Analysis of Social Media: A Systematic Overview of Automated Methods. *The ANNALS of the American Academy of Political and Social Science*, *659*(1), 78–94. http://doi.org/10.1177/0002716215569197

Sjøberg, D. I. K., Dybå, T., & Jørgensen, M. (2007). The Future of Empirical Methods in Software Engineering Research. *Future of Software Engineering*, *SE-13*(1325), 358–378. http://doi.org/10.1109/FOSE.2007.30

Slack, N., Brandon-Jones, A., Johnston, R., & Betts, A. (n.d.). *Operations and process management : principles and practice for strategic impact*.

Sletholt, M. T., Hannay, J., Pfahl, D., Benestad, H. C., & Langtangen, H. P. (2011). A Literature Review of Agile Practices and Their Effects in Scientific Software Development. *Proceeding of the 4th International Workshop on Software Engineering for Computational Science and Engineering - SECSE '11*, 1. http://doi.org/10.1145/1985782.1985784

Strode, D. E. (2015). A dependency taxonomy for agile software development projects. *Information Systems Frontiers*, *18*(1), 23–46. http://doi.org/10.1007/s10796-015-9574-1

Strode, D. E., Huff, S. L., Hope, B., & Link, S. (2012). Coordination in co-located agile software development projects. *Journal of Systems and Software*, *85*(6), 1222–1238. http://doi.org/10.1016/j.jss.2012.02.017

Swartout, P. (2012). *Continuous Delivery and DevOps: A Quickstart guide*. *O'Reilly Media*. Retrieved from http://shop.oreilly.com/product/9781849693684.do

Wang, X., Conboy, K., & Cawley, O. (2012). "Leagile" software development: An experience report analysis of the application of lean approaches in agile software development. *Journal of Systems and Software*, *85*(6), 1287–1299. http://doi.org/10.1016/j.jss.2012.01.061

Womack, J. P., & Jones, D. T. (1996). Lean Thinking: Banish Waste and Create Wealth in Your Corporation, Revised and Updated: James P. Womack, Daniel T. Jones. Retrieved April 12, 2016, from http://www.amazon.com/Lean-Thinking-Corporation-Revised-Updated/dp/0743249275/ref=asap_bc?ie=UTF8

Womack, J. P., Jones, D. T., & Roos, D. (1990). The Machine That Changed the World: The Story of Lean Production-- Toyota's Secret Weapon in the Global Car Wars That Is Now Revolutionizing World Industry. Retrieved April 12, 2016, from http://www.amazon.com/Machine-That-Changed-World-Revolutionizing/dp/0743299795/ref=asap_bc?ie=UTF8

Yin, R. K. (2009). *Case Study Research: Design and Methods*. (L. Bickman & D. J. Rog, Eds.)*Essential guide to qualitative methods in organizational research* (Vol. 5). Sage Publications. http://doi.org/10.1097/FCH.0b013e31822dda9e

Yu, X., & Petter, S. (2014). Understanding agile software development practices using shared mental models theory. *Information and Software Technology*, *56*(8), 911–921. http://doi.org/10.1016/j.infsof.2014.02.010

7.2: Figures

7.2.1: List of figures

Included within the text document:

Included within the text document:

Referred to in the appendix throughout the text document:

Figure 35: Findings I3 Consult. Own creation.

Figure 36: Findings Brainbot. Own Creation.

Figure 37: Findings Siemens. Own creation.

7.2.2: Figures not included in text

Figure 17: Search phrases and results. Own creation

Lean Search Phrases	Agile Search Phrases
Review = 8	Review = 9
Thinking = 24	Thinking = 3
Principles = 9	Principles = 6
Tools = 4	Tools = 1
Practices = 11	Practices = 9
Development =47	Development = 121
Product Development = 25	Product Development = 7
Project Management = 5	Project Management = 19
Software Development = 8	Software Development = 79

Figure 18: LinkedIn review search findings. Own creation.

Level of abstraction	Search Phrase	Group #1	Group #2
2.0 Practices	Scrum	96	30
	Kanban	14	3
	Feature	6	3
	Requirements	7	5
	DevOps	3	19
	Start-up	2	0
	Flow	2	0
	Scrumban	1	0
	XP	0	0
3.0 Tools, Activities & Metrics	Sprint	56	0
	Tools	9	5
	Activities	7	1
	Iteration	6	0
	Velocity	5	0
	Metrics	1	4

Figure 19: Selected literature on Lean, Agile and Combined. Own creation.

Selected Lean literature	Topic	Field of Study
(1990) Womack, Jones & Ross	Foundation & principles	Case study on Toyota
(1996) Womack, Jones & Ross	Foundation & principles	Conceptualization of Toyota Case
(1996) Karlsson & Åhlstrøm	In all processes	Developing operationalized model
(2003) Poppendieck	Software development	Holistic multi-case study
(2006) Liker & Morgan	Product development	Case study on Toyota
(2008) Ries	Start-up development	Developing a Lean Start-up framework
(2010) Poppendieck	Software development	Case study & best practice framework
(2011) Hoppmann, Rebentisch et al.	Product development	Framework for organizing Lean development
(2012) Modig & Åhlström	Foundation & principles	Multi-case & Literature study
(2012) Pillai, Pundir & Ganapathy	Software development	Case study on Six Sigma development
(2012) Mueller & Thoring	Start-up development	Comparing User-Driven methods
(2013) Pernstål, Feldt & Gorschek	Software development	Literature review of large-scale projects
(2014) Power & Conboy	Software development	Literature review on software waste
(2015) León & Farris	Product development	Literature review on development
(2015) Kobus & Westner	IT organizations	Literature review on IT organizations

Selected Agile literature	Topic	Field of Study
(2002) Cockburn	Foundation & principles	Multi-case study & Literature view
(2002) Highsmith	Foundation & principles	In-depth interviews with Agile founders
(2008) Dybå & Dingsøyr	Principles & Context	Literature review on empirical studies
(2008) Pikkarainen, Haikara et al.	Project co-ordination	Embedded case study on project teams
(2011) Sletholt, Hannay et al.	Practices & effects	Observations & Interviews on practices

(2012) Strode, Huff, Hope et al.	Practices & effects	Multi-case study on Co-Location
(2012) Dingsøyr, Nerur et al.	Foundation & principles	Literature review on concept & contributors
(2012) Moe, Aurum & Dybå	Project co-ordination	Multi-case study on shared decision-making
(2013) Javdani, Zulzalil & Ghani	Project Metrics	Literature study & best practice
(2014) Salah, Paige & Cairns	Project co-ordination	Literature review on user integration
(2014) Yu & Petter	Practices & challenges	Conceptual analysis through 'shared values'
(2014) Brhel, Manuel et al.	Principles & practices	Literature review on user integration
(2015) Strode	Project co-ordination	Multi-case study on project dependency
(2015) Inayat, Salim et al.	Practices & challenges	Literature review on requirements engineering
(2015) Rizvi, Bagheri et al.	Practices & challenges	Literature review on virtual collaboration

Selected combined literature	**Topic**	**Field of Study**
(2012) Rodríguez, Markkula et al.	Principles & Practices	Survey on practices in Finland
(2012) Wang, Conboy & Cawley	Principles & Practices	Conceptual analysis on experience reports
(2014) Fagerholm & Pagels	Principles & Practices	Survey on perceptions on concepts
(2014) Floris, Amrit & Daneva	Concepts in DevOps	Literature review on DevOps collaboration
(2014) Dharmapal & Sikamani	Scrum development	Conceptual analysis on Literature studies
(2014) Al-Baik & James	Kanban Methodology	Literature review on Kanban approach
(2015) Kupiainen, Mäntylä et al.	Performance metrics	Literature review on most frequent usages
(2016) Reddy	Scrumban	Multiple-case study on Scrumban

Figure 20: The Lean House (Karlsson & Åhlström, 1996).

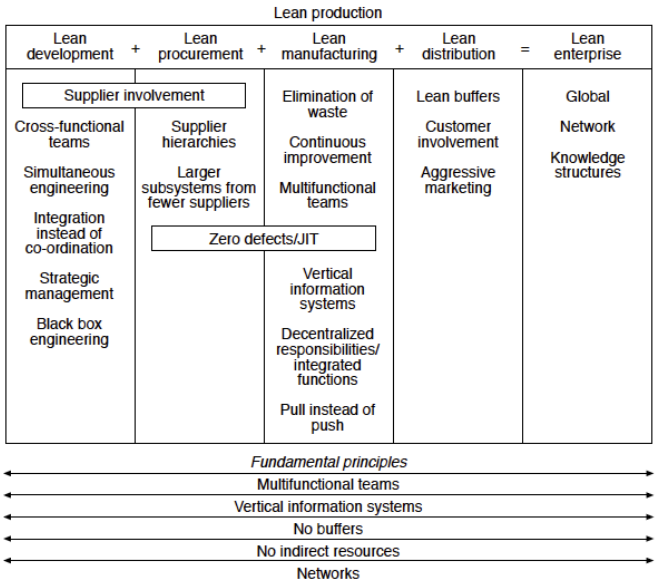


Figure 21: Resource and flow efficiency (Modig & Ahlstrom, 2012).

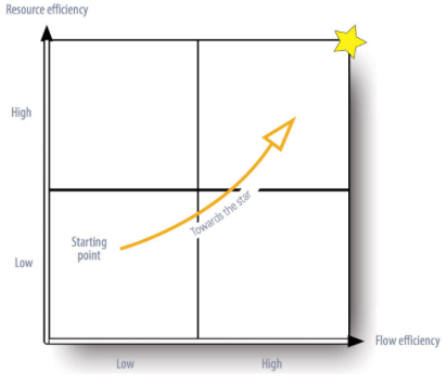


Figure 22: The difference between push and pull systems (Slack, Brandon-Jones, Johnston, & Betts 2015).

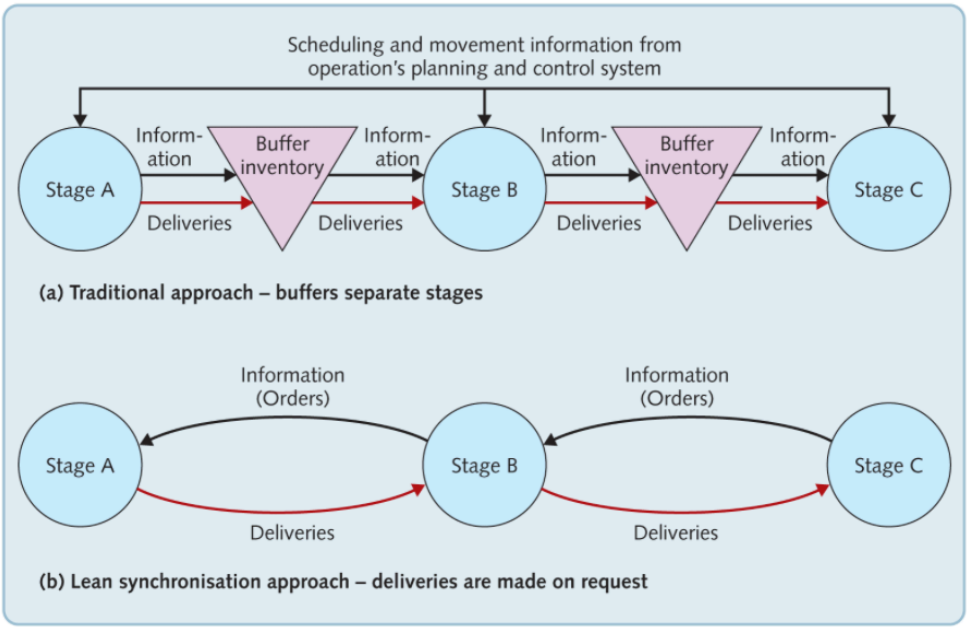


Figure 23: Lean product development and production (León & Farris, 2011).

		People	Process		Tools & Techniques	
LPD frameworks	**LPD subsystems** *(Liker and Morgan, 2006)*					
	Lean Design Techniques *(Womack et al., 1991)*	Leadership, Team-work	Communication		Simultaneous Engineering	
	LPD techniques *(Karlsson and Ahlström, 1996)*	Cross-functional teams	Strategy	Supplier Involvement	Simultaneous (concurrent) Engineering	
	LPD principles *(Ward, 2007)*	ESD, Teams of Responsible Experts	Value focus		SBCE	Cadence Pull, & Flow
PD framework	**PD management domains** *(Kahn et al., 2006)*	People	Strategy, Portfolio Management, Market research	Process		Performance evaluation
LPD within PD framework	**LPD knowledge domains** *(This research)*	Knowledge-based networks	Strategy, Decisions	Supplier/ Partnership	Process modeling	Lean Mfg. principles, Performance

Figure 24: Overview of user-input methods (Cooper & Dreher, 2010)

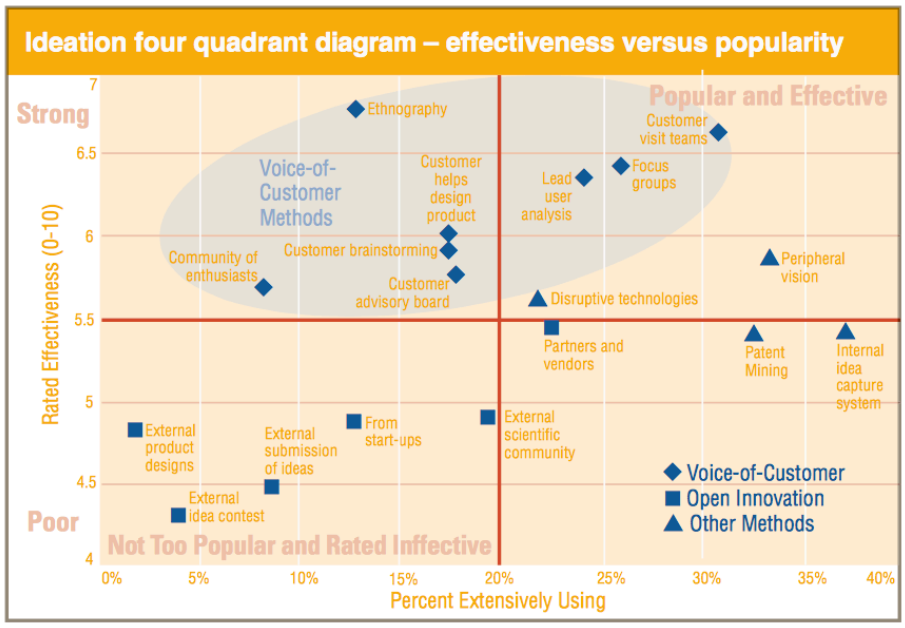


Figure 25: Lean Development and Risk Entrepreneurship (Highsmith, 2002)

Figure 26: Moving towards a focus on impediments to flow (Power & Conboy, 2014)

Category	Definition	How it Impedes Flow
Extra Features	Extra Features are those features that are added without either a proven need or valid hypothesis.	Extra features impede the flow of valuable work through the system by consuming time and effort that could otherwise be spent on more value-adding work. They later prove to add no value for customers, or delay the delivery of more valuable features.
Delays	A delay is a situation in which something happens later than it should, and implies a holding back, usually by interference, from completion or arrival.	Delays impede the flow of work through the system by adding to the overall lead time from request or idea to delivered product or service.
Handovers	Handovers occur whenever incomplete work must be handed over from one person or group to another.	Handovers impede the flow of work through the system by adding delays, requiring more people, or losing knowledge as work is handed over from one person or group to another.
Failure Demand	Failure demand refers to the demand placed on systems (including teams and organizations) and is "demand caused by a failure to do something or do something right for the customer"	It impedes flow by consuming time and effort that could be spent on value-adding work.
Work In Progress	Work in progress is analogous to inventory in software development. It is work that is not yet complete, and, therefore, does not yet provide any value to the business or the customer.	Too much work in progress impedes the flow of work through the system by slowing down the flow of work for individual work items, and delaying the point at which value can be realized.
Context Switching	Context switching occurs when people or teams divide their attention between more than one activity at a time	Context switching impedes the flow of work through the system by adding to the overall lead time from request or idea to delivered product or service, and by causing failure demand and relearning.
Unnecessary Motion	Unnecessary motion is any movement of people, work or knowledge that is avoidable, that impedes the smooth flow of work, or that creates additional inefficiencies	Unnecessary motion impedes the flow of work through the system by adding overhead and causing delays in information or decision-making
Extra Processes	Extra processes generate extra work that consumes time and effort without adding value	Extra processes impede the flow of work through the system by adding additional or incorrect/unsuitable activities
Unmet Human Potential	Unmet human potential is the waste of not using or fostering people's skills and abilities to their full potential	Generally there is an opportunity cost through failing to reach the potential capability of the system. The flow of work, and the associated value, is neither as effective nor efficient as it could be.

Figure 27: When is it appropriate to use Lean Six Sigma? (Pillai et al., 2012).

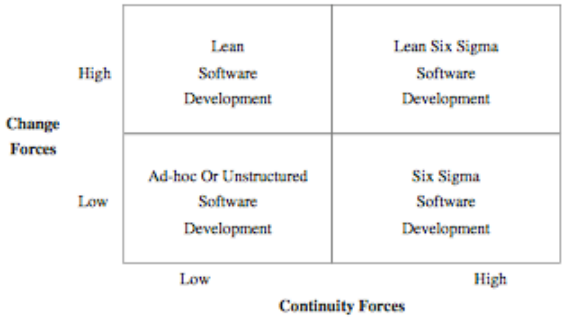


Figure 28: Traditional vs Agile software development (Dybå & Dingsøyr, 2008).

	Traditional development	Agile development
Fundamental assumption	Systems are fully specifiable, predictable, and are built through meticulous and extensive planning	High-quality adaptive software is developed by small teams using the principles of continuous design improvement and testing based on rapid feedback and change
Management style	Command and control	Leadership and collaboration
Knowledge management	Explicit	Tacit
Communication	Formal	Informal
Development model	Life-cycle model (waterfall, spiral or some variation)	The evolutionary-delivery model
Desired organizational form/structure	Mechanistic (bureaucratic with high formalization), aimed at large organizations	Organic (flexible and participative encouraging cooperative social action), aimed at small and medium-sized organizations
Quality control	Heavy planning and strict control. Late, heavy testing	Continuous control of requirements, design and solutions. Continuous testing

Figure 29: Introduction to Agile practices (Dybå & Dingsøyr, 2008.

Agile method	Description
Crystal methodologies	A family of methods for co-located teams of different sizes and criticality: Clear, Yellow, Orange, Red, Blue. The most agile method, Crystal Clear, focuses on communication in small teams developing software that is not life-critical. Clear development has seven characteristics: frequent delivery, reflective improvement, osmotic communication, personal safety, focus, easy access to expert users, and requirements for the technical environment
Dynamic software development method (DSDM)	Divides projects in three phases: pre-project, project life-cycle, and post project. Nine principles underlie DSDM: user involvement, empowering the project team, frequent delivery, addressing current business needs, iterative and incremental development, allow for reversing changes, high-level scope being fixed before project starts, testing throughout the lifecycle, and efficient and effective communication
Feature-driven development	Combines model-driven and agile development with emphasis on initial object model, division of work in features, and iterative design for each feature. Claims to be suitable for the development of critical systems. An iteration of a feature consists of two phases: design and development
Lean software development	An adaptation of principles from lean production and, in particular, the Toyota production system to software development. Consists of seven principles: eliminate waste, amplify learning, decide as late as possible, deliver as fast as possible, empower the team, build integrity, and see the whole
Scrum	Focuses on project management in situations where it is difficult to plan ahead, with mechanisms for "empirical process control"; where feedback loops constitute the core element. Software is developed by a self-organizing team in increments (called "sprints"), starting with planning and ending with a review. Features to be implemented in the system are registered in a backlog. Then, the product owner decides which backlog items should be developed in the following sprint. Team members coordinate their work in a daily stand-up meeting. One team member, the scrum master, is in charge of solving problems that stop the team from working effectively
Extreme programming (XP; XP2)	Focuses on best practice for development. Consists of twelve practices: the planning game, small releases, metaphor, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, 40-h week, on-site customers, and coding standards. The revised "XP2" consists of the following "primary practices": sit together, whole team, informative workspace, energized work, pair programming, stories, weekly cycle, quarterly cycle, slack, 10-minute build, continuous integration, test-first programming, and incremental design. There are also 11 "corollary practices"

Figure 30: Different purposeful combinations of Lean and Agile (Wang et al., 2012).

Type of lean application		
Non-purposeful combination of agile and lean in software development processes		
Purposeful application of lean approaches in agile software development	Lean approaches are applied to the business areas related to software development	**Agile within, lean out-reach**: using lean approaches to interact with neighbouring business units while keeping agile development processes internally
	Lean approaches are applied to software development processes directly	**Lean facilitating agile adoption** before or during the transition process
		Lean within agile: using various lean elements to improve agile processes
		From agile to lean: comprehensive application of lean approaches to transform agile processes
		Synchronising agile and lean: agile team and lean team work in parallel in a synchronised manner

Figure 31: Previous findings on the relation between Lean and Agile principles (Peterson, 2010)

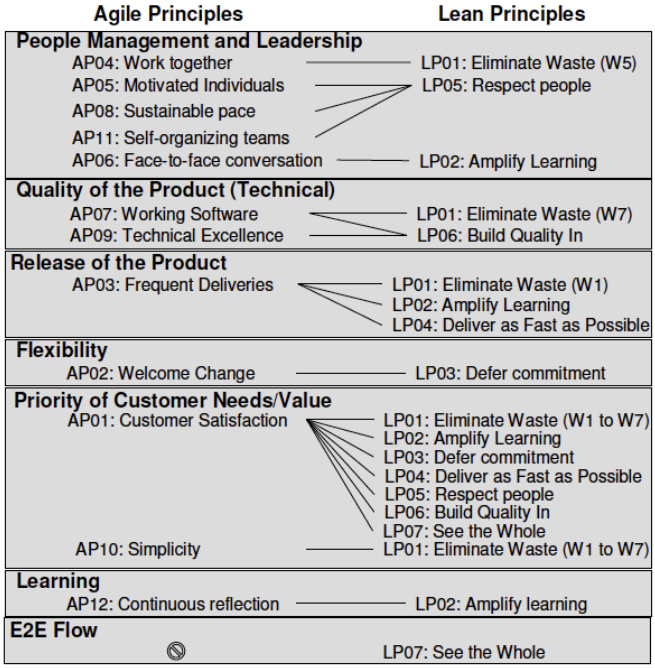


Figure 32: Findings of principles and practices. Own creation

Level of Abstraction	Foundation/Principles	Practices/Schools
1. Lean	(A) Kaizen & Kaikaku	(G) Kanban
	(B) Jidoka	(H) Development
	(C) Zero Defects	(I) Customer Discovery
	(D) Muda/Mura/Muri	(J) MVP
	(E) Value-Stream	(K) Risk Entrepreneurship
	(F) Just-in Time & Pull	(L) Comphrehensive School
2. Agile		
	(A) Individuals and Interactions	(E) Continous Integration
	(B) Working Software	(F) Scrum
	(C) Customer Collaboration	(G) FFD
	(D) Chaordic and Self-organizing	(H) XP
3. Combined		
	(A) Impediments to Flow	(E) Leagile
	(B) Continous Improvement	(F) Selective School
	(C) Continous Delivery	(G) Scrumban
	(D) Continous Deployment	(H) DevOps

Figure 33: Categorization of posts. Own creation

Categorization	Content of post
Scrum	
#2,7,8,16,18,20	How to manage stories within sprints
#9,12,15	Tasks and responsibilities
#4,13	Performance metrics for Scrum
#11	Contact practices within Scrum
Kanban	
#5	How to use the Kanban System
#18	What is the relation between Kanban & Sprints
DevOps	
#3,10	Dividing tasks between Dev & Ops
Planning & motivation	
#14	How to use roadmaps
#1	How to foster creativity and innovation
#6	What is the role of infrastructure and architecture

Figure 34: Findings Vizgu. Own creation.

Level of abstraction	Applied at Vizgu

Principles	Start-up based Agile, Continous Delivery
Practices	Kanban, XP, Continous Integration
Tools	Jira, Skype, Visual Studio, Dropbox, Apple & Google API's
Activities	Feature and design driven splitting, parallel and easy editable programming, monkey testing, no acceptance or done criteria, marketing and development division, ad hoc process evaluation.
Performance Metrics	Ad hoc task estimation & visualization and documentation of needed and completed tasks in virtual Kanban
Co-Location	Only development is co-located

Figure 35: Findings I3 Consult. Own creation.

Level of abstraction	Applied at I3 Consult
Principles	Client based: Agile, Lean & Waterfall
Practices	Waterfall, Scrum & Kanban
Tools	Microsoft Link, Jira Kanban, custom made testing scripts
Activities	Daily stand-up meeting with screens locked on Kanban board. Classic Scrum setup: Shared planning, 3-week sprints. Splitting of stories according to user-stories. Process evaluation after ended sprints. Customized testing scripts for test, media and system environments.
Performance Metrics	WIP, Velocity and Burndown in virtual Jira Kanban. The manager mostly looks into rate of tasks done.
Co-Location	Clients are co-located, co-workers are collaborating virtually

Figure 36: Findings Brainbot. Own Creation.

Level of abstraction	Applied at Brainbot Technologies
Principles	Agile & Lean, Continous delivery
Practices	Elements of Scrum and Continous Integration
Tools	Skype, Slack, Github, Travis, Trello, Zoo
Activities	Daily stand-up meeting, weekly status meetings, Continous integration through Slack, Submitting issues on Github, Customer screening, Peer-review.
Performance Metrics	Proof-of-work in Github repository and client feedback
Co-Location	Team and clients globally spread out

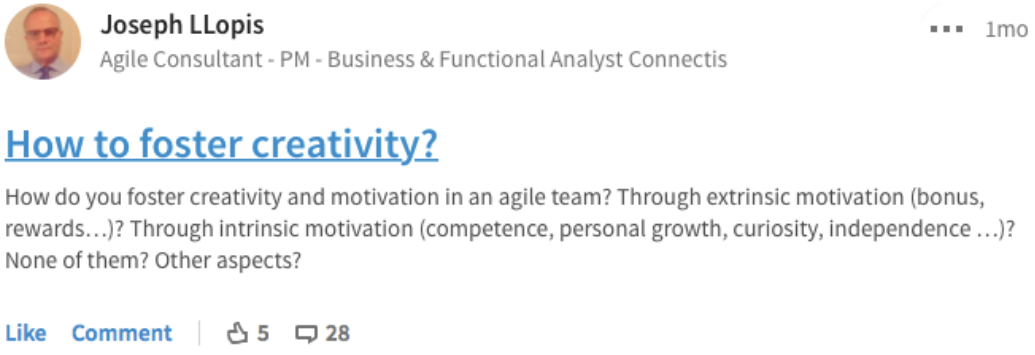
Figure 37: Findings Siemens. Own creation.

Level of abstraction	Siemens employee on boarding project
Principles	Agile, Waterfall & Lean

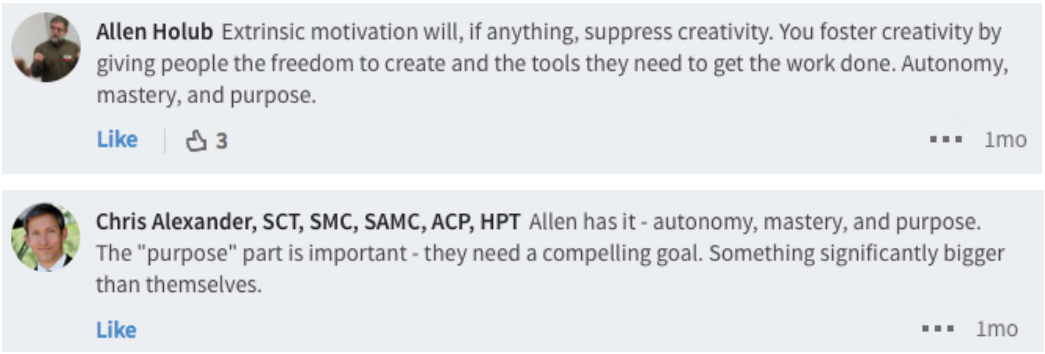
Practices	Kanban, Scrum
Tools	Slack, Soup Kanban, Github, Jenkins, Travis
Activities	Fixed project duration, product vision, structured and continuous user-feedback, ad hoc evaluation, self-organizing of subtasks.
Performance Metrics	Task estimation & project dependent
Co-Location	Relevant departments in same building

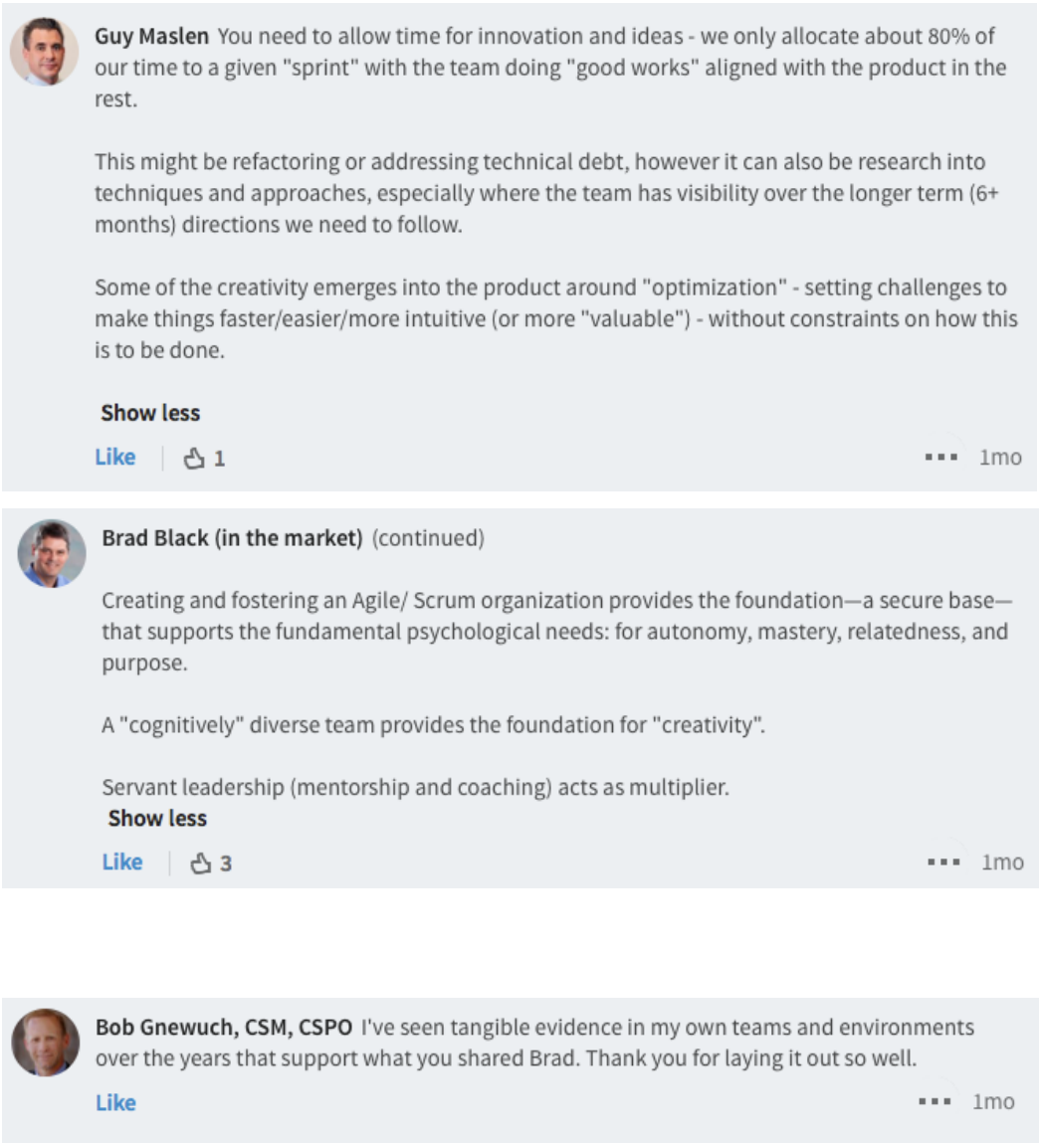
7.3: LinkedIn data

#1.



#Selected comments





Guy Maslen You need to allow time for innovation and ideas - we only allocate about 80% of our time to a given "sprint" with the team doing "good works" aligned with the product in the rest.

This might be refactoring or addressing technical debt, however it can also be research into techniques and approaches, especially where the team has visibility over the longer term (6+ months) directions we need to follow.

Some of the creativity emerges into the product around "optimization" - setting challenges to make things faster/easier/more intuitive (or more "valuable") - without constraints on how this is to be done.

Show less

Like | 1 ··· 1mo

Brad Black (in the market) (continued)

Creating and fostering an Agile/ Scrum organization provides the foundation—a secure base—that supports the fundamental psychological needs: for autonomy, mastery, relatedness, and purpose.

A "cognitively" diverse team provides the foundation for "creativity".

Servant leadership (mentorship and coaching) acts as multiplier.

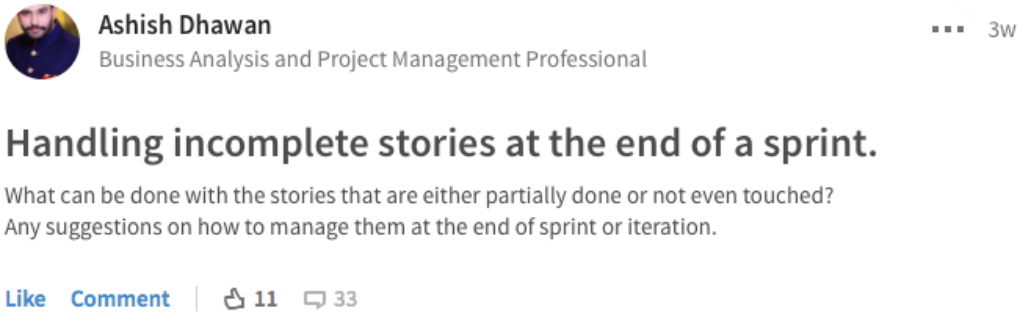
Show less

Like | 3 ··· 1mo

Bob Gnewuch, CSM, CSPO I've seen tangible evidence in my own teams and environments over the years that support what you shared Brad. Thank you for laying it out so well.

Like ··· 1mo

#2.



Ashish Dhawan ··· 3w

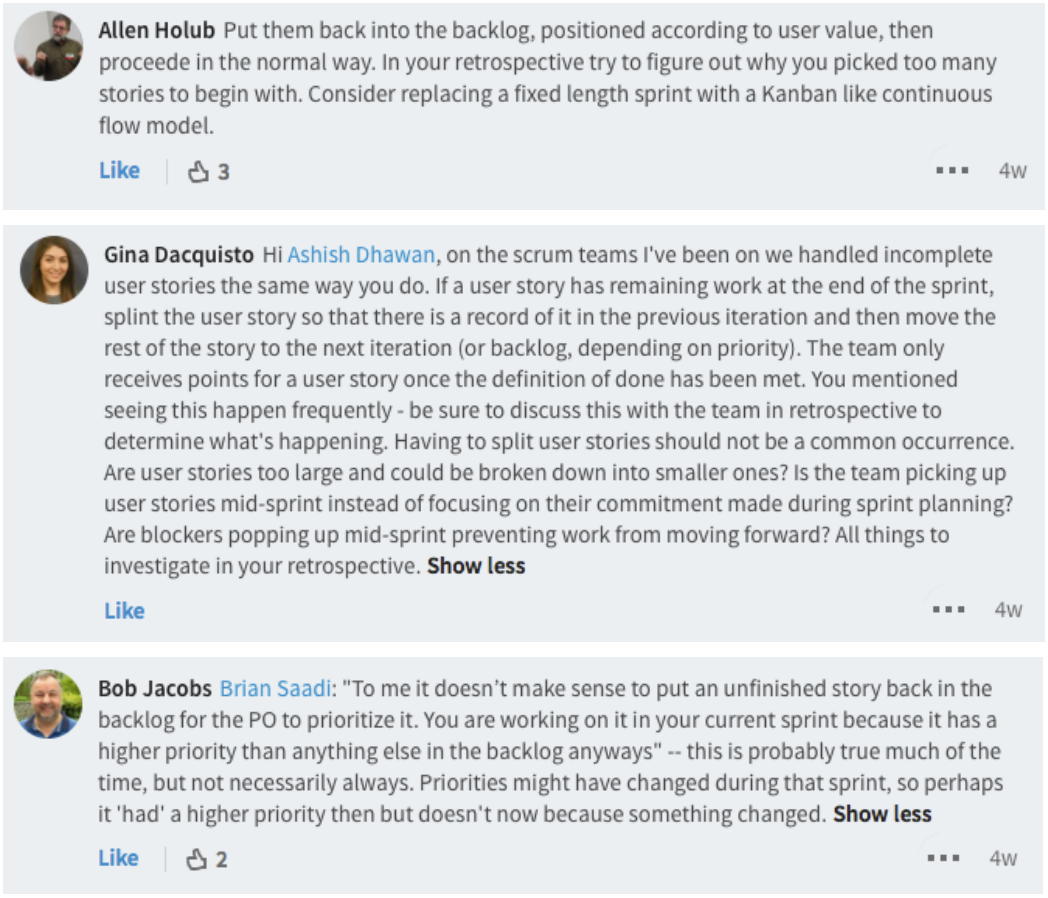
Business Analysis and Project Management Professional

Handling incomplete stories at the end of a sprint.

What can be done with the stories that are either partially done or not even touched?
Any suggestions on how to manage them at the end of sprint or iteration.

Like Comment | 11 33

#Selected comments



Allen Holub Put them back into the backlog, positioned according to user value, then proceede in the normal way. In your retrospective try to figure out why you picked too many stories to begin with. Consider replacing a fixed length sprint with a Kanban like continuous flow model.

Like | 3 ··· 4w

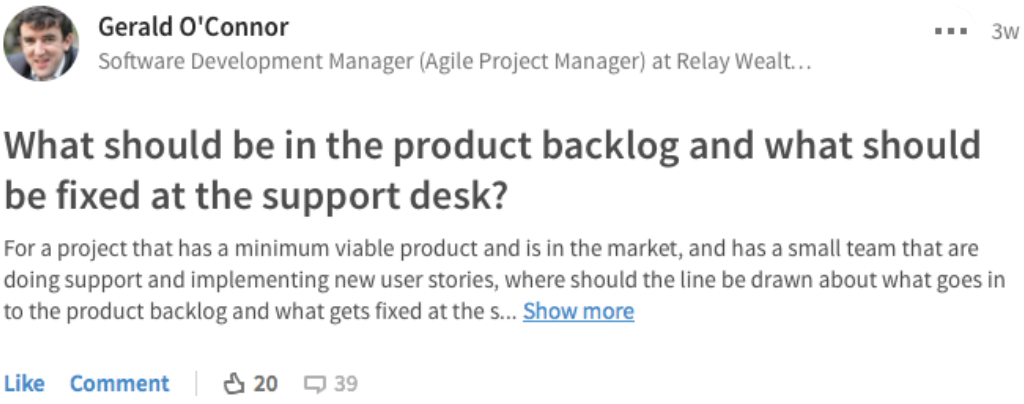
Gina Dacquisto Hi Ashish Dhawan, on the scrum teams I've been on we handled incomplete user stories the same way you do. If a user story has remaining work at the end of the sprint, splint the user story so that there is a record of it in the previous iteration and then move the rest of the story to the next iteration (or backlog, depending on priority). The team only receives points for a user story once the definition of done has been met. You mentioned seeing this happen frequently - be sure to discuss this with the team in retrospective to determine what's happening. Having to split user stories should not be a common occurrence. Are user stories too large and could be broken down into smaller ones? Is the team picking up user stories mid-sprint instead of focusing on their commitment made during sprint planning? Are blockers popping up mid-sprint preventing work from moving forward? All things to investigate in your retrospective. **Show less**

Like ··· 4w

Bob Jacobs Brian Saadi: "To me it doesn't make sense to put an unfinished story back in the backlog for the PO to prioritize it. You are working on it in your current sprint because it has a higher priority than anything else in the backlog anyways" -- this is probably true much of the time, but not necessarily always. Priorities might have changed during that sprint, so perhaps it 'had' a higher priority then but doesn't now because something changed. **Show less**

Like | 2 ··· 4w

#3.



Gerald O'Connor ··· 3w

Software Development Manager (Agile Project Manager) at Relay Wealt…

What should be in the product backlog and what should be fixed at the support desk?

For a project that has a minimum viable product and is in the market, and has a small team that are doing support and implementing new user stories, where should the line be drawn about what goes in to the product backlog and what gets fixed at the s... Show more

Like Comment | 20 39

#Selected comments

Sion Evans I recommend using a category on the product backlog to differentiate between 'product development' and 'support' items. You will then be able to "filter" when discussing with the interested parties.

Apply consistent use of prioritisation, driven by what impacts the business.

A good product owner should be able to recognise that support & product development items are both important.

Sprint capacity should allow of number of points for support items, which can be flexed up or down depending on how stable the product is. **Show less**

Cheryl Ash Support requests such as installation or requests for information are going to impact feature delivery if one team is responsible for both. However, if there are a lot of bugs being introduced that might signal a problem. The team should discuss this and perhaps adjust the definition of done.

I would suggest having two teams. A dedicated support team that probably works in some form of Kanban and a feature development team.

When features are released it could then become the responsibility of the support team to take care of related requests.

Where you record features doesn't really matter. You could keep them as part of the same backlog or in a separate system. I've used zendesk in the past for support teams.

Show less

David Cooper At an early stage of a product it is great to have developers involved in support. I have found that when a support issue is a developer's problem they will do some development so they never have to deal with it again! While a support person will provide a work around and be happy to keep solving the same problem over and over again (job security?). Also agree hugely with Cheryl, lots of bugs are a sign of a problem with code or process that needs fixing. If we have more than 2 bugs introduced in a release we have a meeting to find out why this happened and what we need to put in place so they don't happen again. We are a hard-core Continuous Improvement team. At the MVP stage it is critical that the whole team be involved.
Show less

Like | 3 ··· 4w

#4.

Ishita Trivedi ··· 3w
Lead Agile Business Analyst | Digital Projects

Story pointing v/s Task estimation

I have come across this question a few times by my agile teams, do we need both story points to measure team velocity and task estimation to gauge actual time spent on each task within story?

#Selected comments

Vishal Chaphekar The number of user stories in a sprint depends upon the total User Points that the team can deliver (that is their current productivity) and should never be related to hours since cost/hours are fixed in agile what changes in the number of user stories delivered and this is very critical when the priority of a userstory changes mid sprint and you need to tradeoff userstories which you would do it based off user story points.
The task estimations and tracking them is important too for a scrum master to know how the user story is coming along, this can be used for burn down charts and combining that with Kanban to minimize the WiP is a great way to reduce sprint risks and wastes. **Show less**

Like ··· 4w

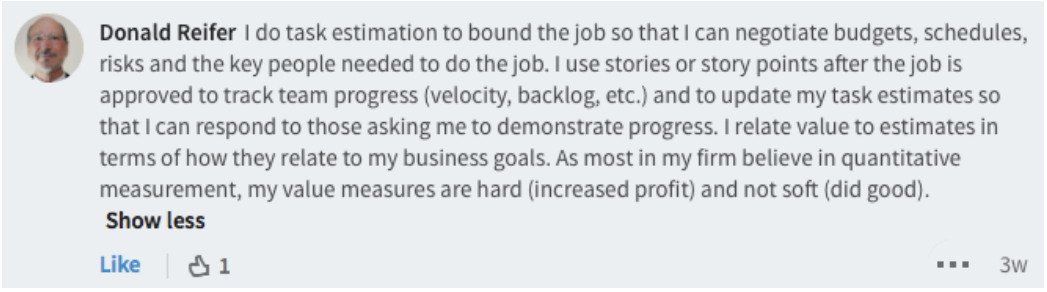
Kevin Kriner It's up to the team! Both are not necessary. I often suggest that teams start with story points only. Then, if the team wants to add tasks, adding tasks without estimates. Then, if the team wants to add estimates to those tasks, adding estimates to the tasks. Teams working in an agile manner can function very well with only story points and velocity (as a ballpark measure of capacity used to help understand approximately how much work a team can deliver in a timeframe or approximately how long it will take the team to deliver a scope of work). **Show less**

Like | 2 ··· 4w

Donald Reifer @Aarte. Velocity used at the team level for internal consumption. Rate of progress used at the subsystem level, when appropriate. Earned value used at the project level, when larger than one or two teams. I provided references on each of these including earned value in earlier posts.

@Paul. Estimate at the project level which develops effort and duration estimates from team inputs. As team inputs varies as a function of size (seen 3 to 30 person teams - we use 3 to 8 internally), this can vary. Internal projects are at most 20 people or 2 to 3 teams. We do not estimate half day tasks. We look at big ones (team of 3 to 8 for 4 to 6 weeks) . This helps with the uncertainty and risk. So we gather the team inputs, sum up their estimates and then synch their schedules , sometimes seeing if we can do them in parallel (look at critical path), and come up with duration and then staff-load. External ones that I work on can be several hundred. We use scaled version of above. **Show less**

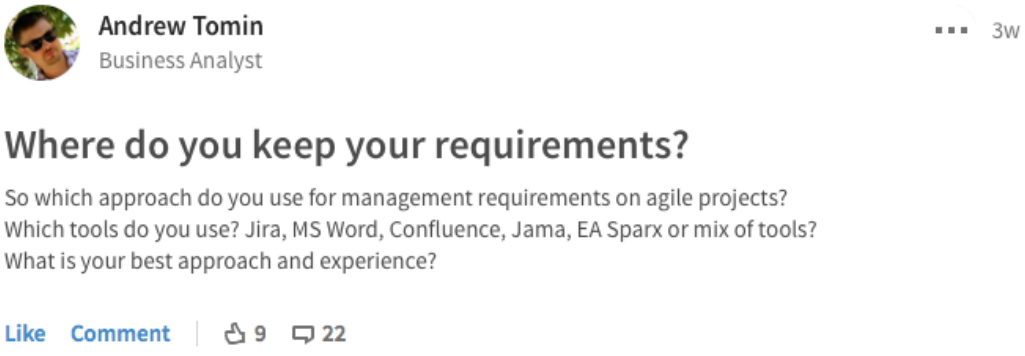
Like | 1 ··· 3w



Donald Reifer I do task estimation to bound the job so that I can negotiate budgets, schedules, risks and the key people needed to do the job. I use stories or story points after the job is approved to track team progress (velocity, backlog, etc.) and to update my task estimates so that I can respond to those asking me to demonstrate progress. I relate value to estimates in terms of how they relate to my business goals. As most in my firm believe in quantitative measurement, my value measures are hard (increased profit) and not soft (did good).
Show less

Like | 1 ··· 3w

#5.



Andrew Tomin
Business Analyst

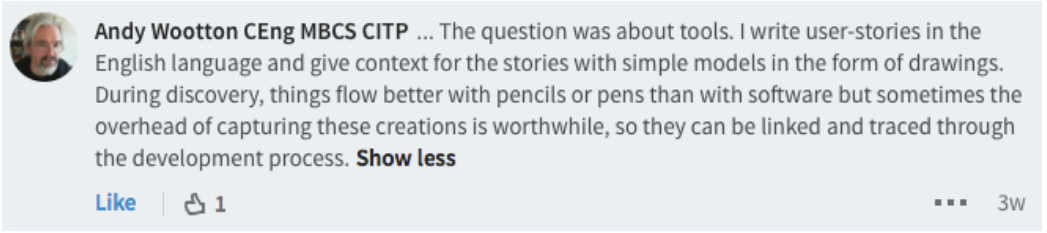
··· 3w

Where do you keep your requirements?

So which approach do you use for management requirements on agile projects?
Which tools do you use? Jira, MS Word, Confluence, Jama, EA Sparx or mix of tools?
What is your best approach and experience?

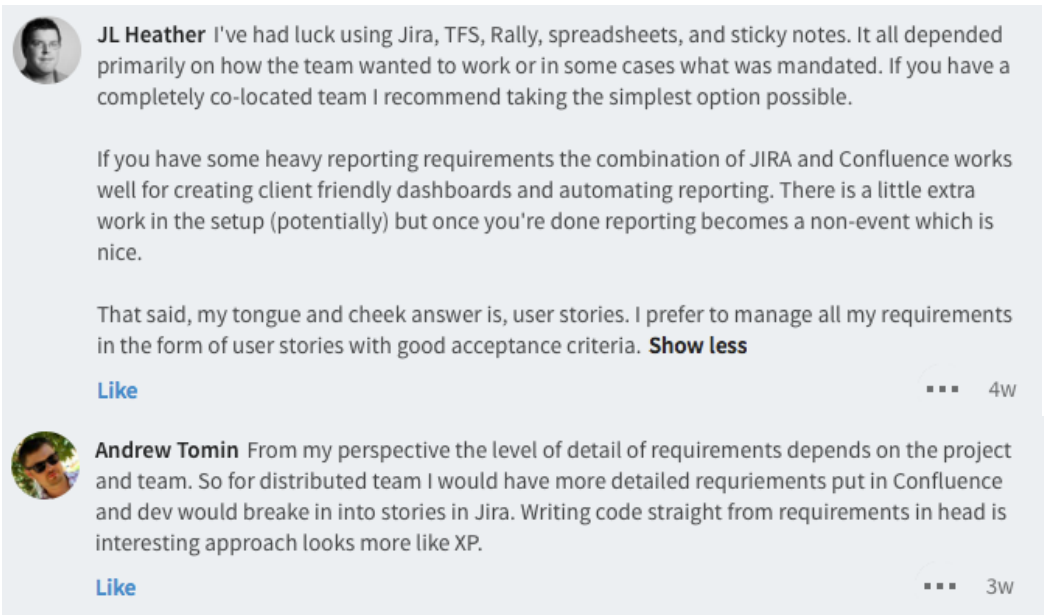
Like Comment | 9 22

#Selected comments



Andy Wootton CEng MBCS CITP ... The question was about tools. I write user-stories in the English language and give context for the stories with simple models in the form of drawings. During discovery, things flow better with pencils or pens than with software but sometimes the overhead of capturing these creations is worthwhile, so they can be linked and traced through the development process. **Show less**

Like | 1 ··· 3w



JL Heather I've had luck using Jira, TFS, Rally, spreadsheets, and sticky notes. It all depended primarily on how the team wanted to work or in some cases what was mandated. If you have a completely co-located team I recommend taking the simplest option possible.

If you have some heavy reporting requirements the combination of JIRA and Confluence works well for creating client friendly dashboards and automating reporting. There is a little extra work in the setup (potentially) but once you're done reporting becomes a non-event which is nice.

That said, my tongue and cheek answer is, user stories. I prefer to manage all my requirements in the form of user stories with good acceptance criteria. **Show less**

Like ··· 4w

Andrew Tomin From my perspective the level of detail of requirements depends on the project and team. So for distributed team I would have more detailed requriements put in Confluence and dev would breake in into stories in Jira. Writing code straight from requirements in head is interesting approach looks more like XP.

Like ··· 3w

Tushar Jain Product Backlog is the list of requirements. Depending upon POs discretion, product back log items are detailed with help of stakeholders and devTeam. During development in close collaboration of devTeam and PO requirements get converted into code.

I got chance to work with sticky notes, spread sheets, word, Rally, Jira, and, version one.

There is no best approach, all depends upon context.
Show less

Like | 2 ··· 4w

#6.

Rich Gould
IT / Supply Chain Leader

··· 3w

Infrastructure/Architecture & Agile

Ok, I'm just beginning to learn about Agile and I've just started an online course about Agile Project Management. It strikes me in what I've seen so far that delivered features seem to assume Infrastructure/Architecture designs are in place and you use Agile to deliver business functional features. Am I understanding this correctly or are Infrastructure deliverables just features in the backlog that are delivered within typical sprint processes? **Show less**

Like Comment | 5 29

#Selected comments

Eric Tucker, CSP, PMI-ACP, CSM, CSPO Infrastructure / architecture changes based on feature needs are commonplace in complex software development. If the team you serve has the capability to make changes at that level, great. If not, there are a few options. 1) let the team experiment in a safe environment where they can't crash the entire infrastructure. 2) bring in a SME to consult with the team and help them acquire the knowledge they need to make infrastructure level changes. 3) if a different team handles infrastructure level work, encourage some inter-team pairing. The point is learning. The more a cross functional team knows, the more value they can add to the business. **Show less**

Like ··· 3w

Chris Alexander, SCT, SMC, SAMC, ACP, HPT Architecture is one aspect of a deliverable feature - not separate. This reflects the concept of vertical slicing and delivering functionality in increments, vice structuring to deliver pieces of code chronologically in layers (ie., design first, then build the back end, then start working on the middle tier...).

If the architecture has broader implications that need to be addressed up front, then "simplicity" is important. Get the minimum necessary information agreed, move forward from there, refine as you go. **Show less**

Like | 3 ··· 3w

#7.

Long Truong Quang Binh
Engineering Manager at CSC Vietnam

3w

Is it possible to add more work to a current sprint?

Hello All,

Assume that the team works in a four-week sprint. After the second week, the team finishes its sprint backlog. Now, the team still has two more weeks. This issue may happen in some very first sprints as the team does not have a correct u... **Show more**

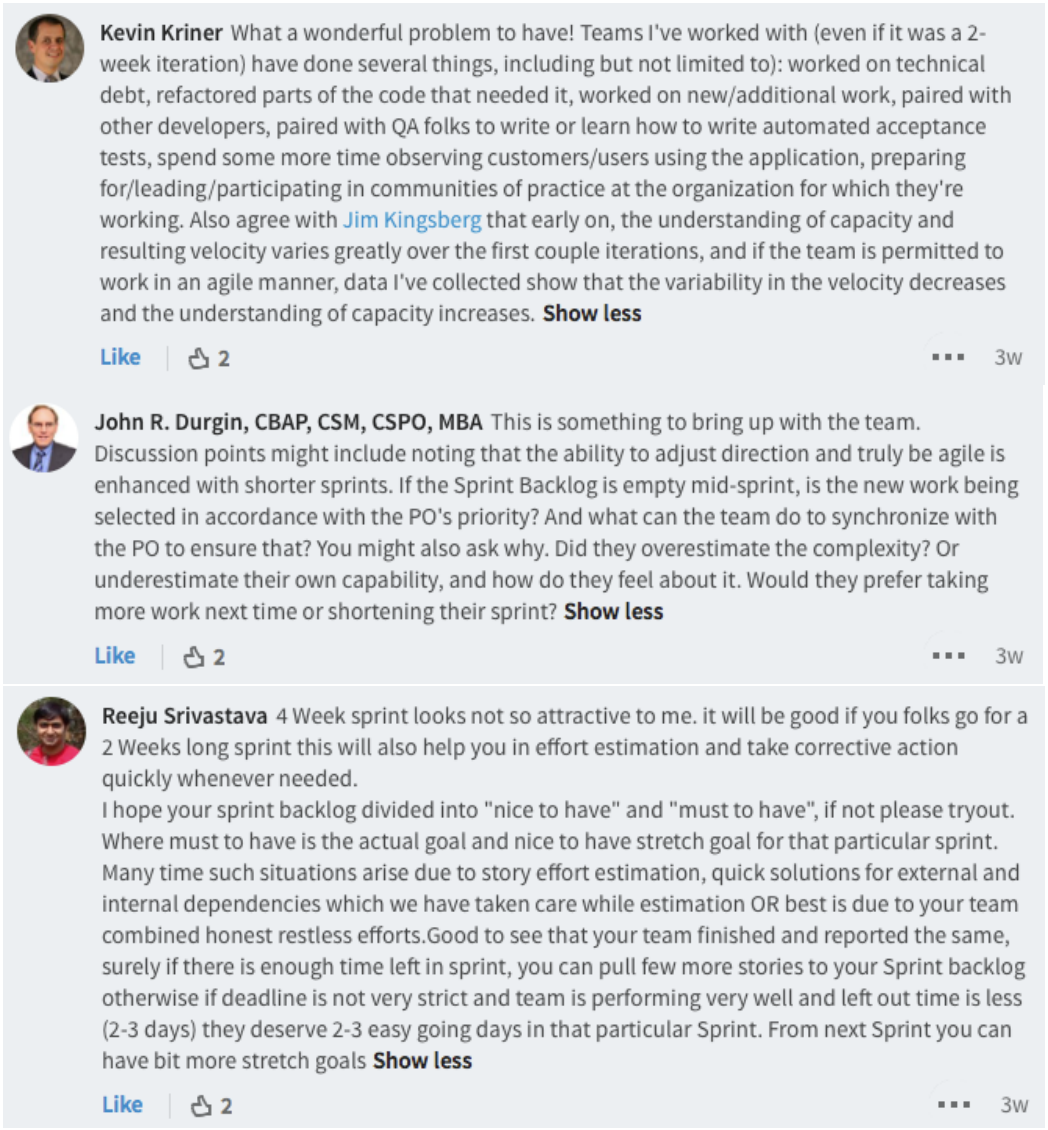
Like Comment | 15 59

#Selected comments

Pete Morris, PMP, PMI-ACP Project estimates, whether from the traditional or Agile world, are expected to be inaccurate because they are based on ever-changing assumptions. Moreover, the further ahead you estimate, the more the accuracy of the estimates decrease. It makes sense then to mitigate the risk of inaccuracy by limiting the time period of the estimate, and to reduce a large complex task into smaller, simpler components that are more realistically estimated. **Show less**

Like | 1

3w



Kevin Kriner What a wonderful problem to have! Teams I've worked with (even if it was a 2-week iteration) have done several things, including but not limited to): worked on technical debt, refactored parts of the code that needed it, worked on new/additional work, paired with other developers, paired with QA folks to write or learn how to write automated acceptance tests, spend some more time observing customers/users using the application, preparing for/leading/participating in communities of practice at the organization for which they're working. Also agree with Jim Kingsberg that early on, the understanding of capacity and resulting velocity varies greatly over the first couple iterations, and if the team is permitted to work in an agile manner, data I've collected show that the variability in the velocity decreases and the understanding of capacity increases. **Show less**

Like | 2 ··· 3w

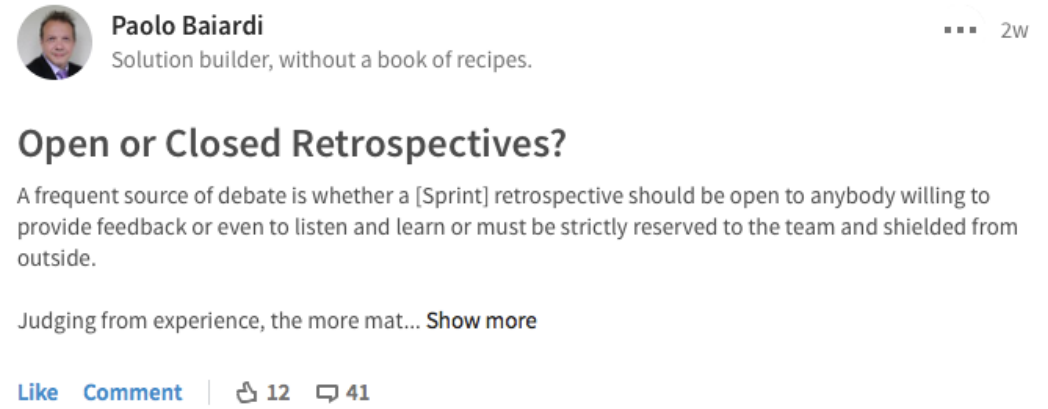
John R. Durgin, CBAP, CSM, CSPO, MBA This is something to bring up with the team. Discussion points might include noting that the ability to adjust direction and truly be agile is enhanced with shorter sprints. If the Sprint Backlog is empty mid-sprint, is the new work being selected in accordance with the PO's priority? And what can the team do to synchronize with the PO to ensure that? You might also ask why. Did they overestimate the complexity? Or underestimate their own capability, and how do they feel about it. Would they prefer taking more work next time or shortening their sprint? **Show less**

Like | 2 ··· 3w

Reeju Srivastava 4 Week sprint looks not so attractive to me. it will be good if you folks go for a 2 Weeks long sprint this will also help you in effort estimation and take corrective action quickly whenever needed.
I hope your sprint backlog divided into "nice to have" and "must to have", if not please tryout. Where must to have is the actual goal and nice to have stretch goal for that particular sprint. Many time such situations arise due to story effort estimation, quick solutions for external and internal dependencies which we have taken care while estimation OR best is due to your team combined honest restless efforts.Good to see that your team finished and reported the same, surely if there is enough time left in sprint, you can pull few more stories to your Sprint backlog otherwise if deadline is not very strict and team is performing very well and left out time is less (2-3 days) they deserve 2-3 easy going days in that particular Sprint. From next Sprint you can have bit more stretch goals **Show less**

Like | 2 ··· 3w

#8.



Paolo Baiardi ··· 2w
Solution builder, without a book of recipes.

Open or Closed Retrospectives?

A frequent source of debate is whether a [Sprint] retrospective should be open to anybody willing to provide feedback or even to listen and learn or must be strictly reserved to the team and shielded from outside.

Judging from experience, the more mat... Show more

Like Comment | 12 41

#Selected comments

Jim Kingsberg I try to get stakeholder feedback in the sprint demo/review. I do encourage feedback beyond what was delivered (and how). We also look at what the team planned vs what got done.
I like keeping the retro constrained to the team. In some cases, the team might look at stakeholders as their problem solvers; instead of the team solving problems, they rely on people outside the team (and that can certainly be the case). One of the "smells" here is if the SM or PO are taking on problem resolution vs team members who can champion the cause. That said, if the team wants someone outside the team to come to a retro, not a problem. (However, I will caution that I've seen some "managers" exhibit bad behaviors (defensiveness, challenging, disagreeing, etc). **Show less**

Like | 1 ··· 3w

Marcel Blok We do both. It depends on what we want to discuss. The key is that all persons that are involved in the subject that is discussed in the retrospective should be there. Quite often this is just the team. Often the PO is also involved. Sometimes others from the organization. If needed and possible you can include someone from outside the organization.

The main goal of the retrospective is to look back on a subject and see what goes well and what can be changed. All people that are involved should be present for this, as we otherwise make decisions for someone else, or we end up with just a complaining-session on some other department. In the end we should have people that agree and know what to do. **Show less**

Like | 4 ··· 3w

Stephan Kristiansen Both options are possible of course, but it's important understand that the team might not feel free to speak their mind if management or other stakeholders are present. If the team wants to discuss some impediments or something they feel they have no chance to actually improve themselves without outside help, then inviting "outsiders" is a possibility. Generally speaking I would keep the retrospective for the team only, to make sure everyone feels safe enough to speak their mind. Eventually its all about the team and about how they best can use the retrospective to inspect and adapt to improve their performance as a team. **Show less**

Like | 2 ··· 3w

#9.

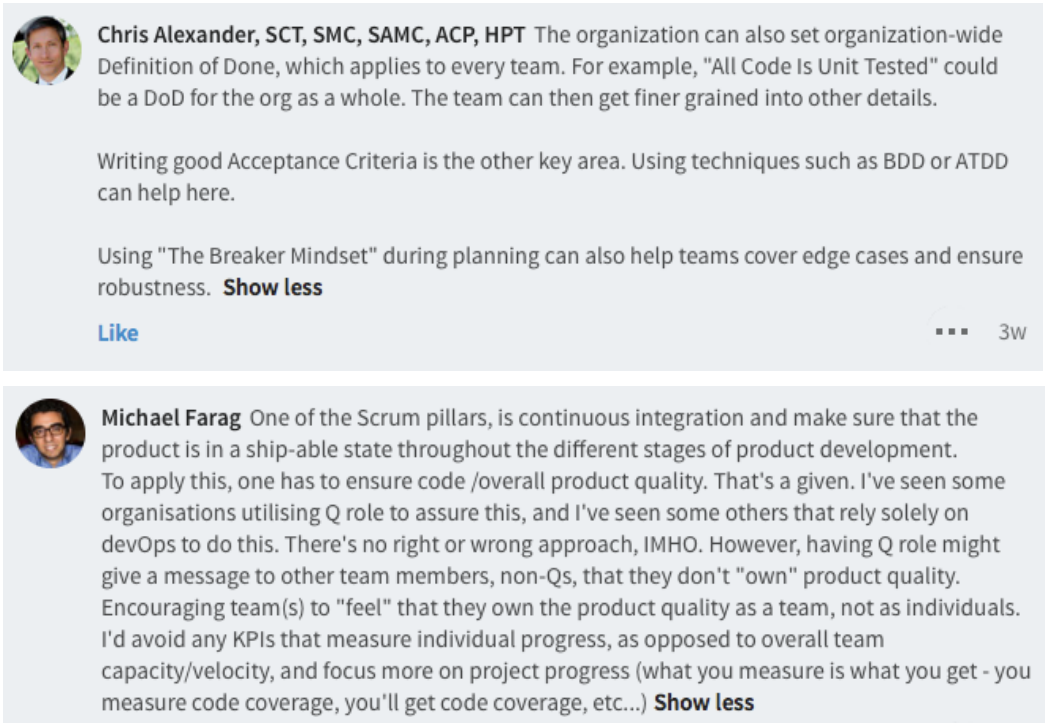
Ankita Sharma ··· 2w
Scrum Master and Quality Consultant at SAP Labs India Pvt Ltd

Quality framework within scrum

Hello, I understand that with scrum the complete responsibility moves to the acrum team. But what kind of quality procedures need to be followed by the team. I am talking in terms of an organization which is moving into acrum from waterfall. Where te... **Show more**

Like Comment | 8 14

#Selected comments



Chris Alexander, SCT, SMC, SAMC, ACP, HPT The organization can also set organization-wide Definition of Done, which applies to every team. For example, "All Code Is Unit Tested" could be a DoD for the org as a whole. The team can then get finer grained into other details.

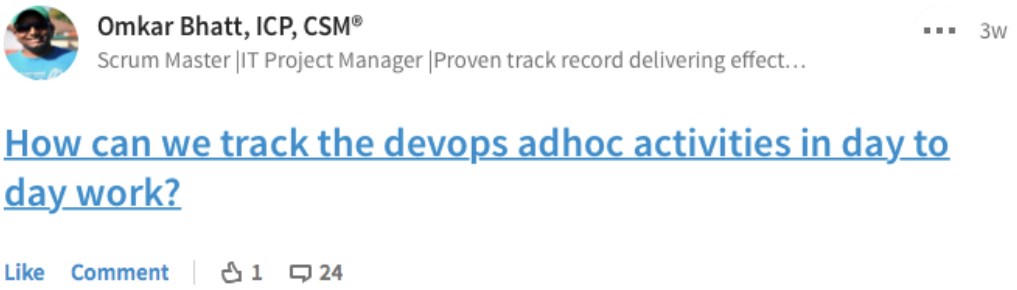
Writing good Acceptance Criteria is the other key area. Using techniques such as BDD or ATDD can help here.

Using "The Breaker Mindset" during planning can also help teams cover edge cases and ensure robustness. **Show less**

Like ··· 3w

Michael Farag One of the Scrum pillars, is continuous integration and make sure that the product is in a ship-able state throughout the different stages of product development.
To apply this, one has to ensure code /overall product quality. That's a given. I've seen some organisations utilising Q role to assure this, and I've seen some others that rely solely on devOps to do this. There's no right or wrong approach, IMHO. However, having Q role might give a message to other team members, non-Qs, that they don't "own" product quality. Encouraging team(s) to "feel" that they own the product quality as a team, not as individuals. I'd avoid any KPIs that measure individual progress, as opposed to overall team capacity/velocity, and focus more on project progress (what you measure is what you get - you measure code coverage, you'll get code coverage, etc...) **Show less**

#10.

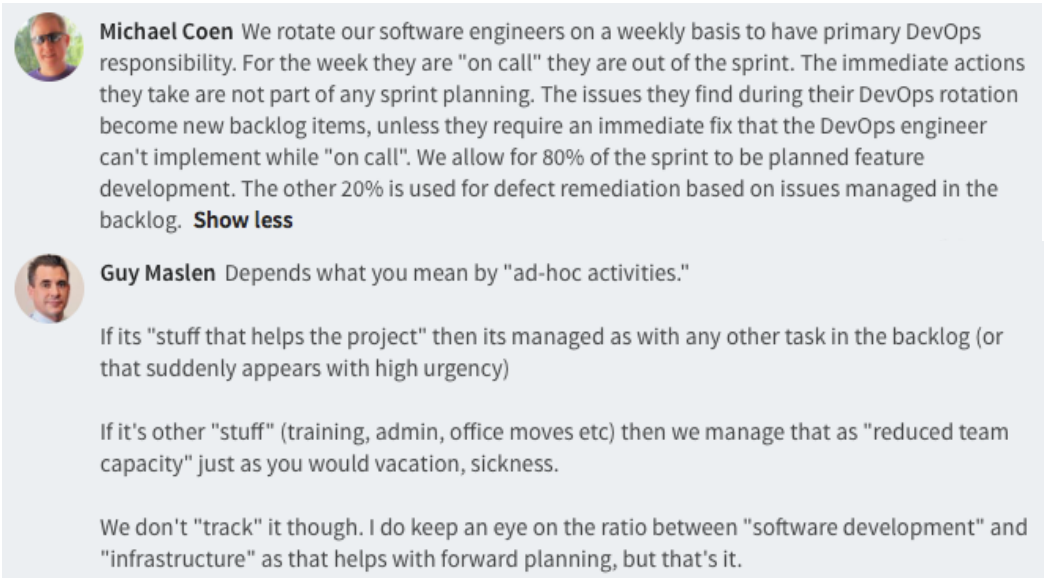


Omkar Bhatt, ICP, CSM® ··· 3w
Scrum Master |IT Project Manager |Proven track record delivering effect…

How can we track the devops adhoc activities in day to day work?

Like Comment | 1 24

#Selected comments



Michael Coen We rotate our software engineers on a weekly basis to have primary DevOps responsibility. For the week they are "on call" they are out of the sprint. The immediate actions they take are not part of any sprint planning. The issues they find during their DevOps rotation become new backlog items, unless they require an immediate fix that the DevOps engineer can't implement while "on call". We allow for 80% of the sprint to be planned feature development. The other 20% is used for defect remediation based on issues managed in the backlog. **Show less**

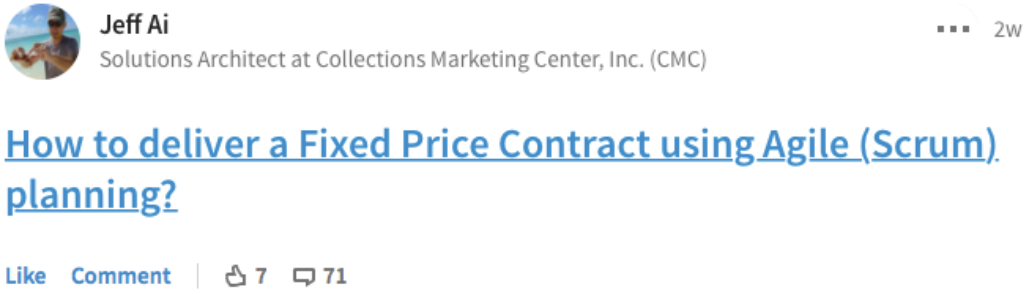
Guy Maslen Depends what you mean by "ad-hoc activities."

If its "stuff that helps the project" then its managed as with any other task in the backlog (or that suddenly appears with high urgency)

If it's other "stuff" (training, admin, office moves etc) then we manage that as "reduced team capacity" just as you would vacation, sickness.

We don't "track" it though. I do keep an eye on the ratio between "software development" and "infrastructure" as that helps with forward planning, but that's it.

#11.

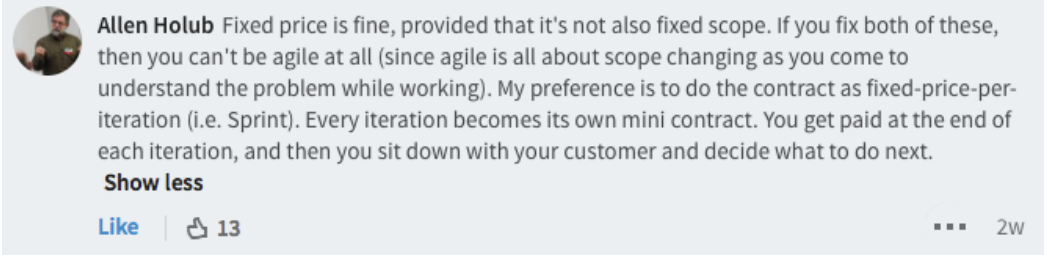


Jeff Ai ··· 2w
Solutions Architect at Collections Marketing Center, Inc. (CMC)

How to deliver a Fixed Price Contract using Agile (Scrum) planning?

Like Comment | 7 71

#Selected comments



Allen Holub Fixed price is fine, provided that it's not also fixed scope. If you fix both of these, then you can't be agile at all (since agile is all about scope changing as you come to understand the problem while working). My preference is to do the contract as fixed-price-per-iteration (i.e. Sprint). Every iteration becomes its own mini contract. You get paid at the end of each iteration, and then you sit down with your customer and decide what to do next.
Show less

Like | 13 ··· 2w

Mahendra Singh Allen, the above approach is a Rapid approach, where continuous collaboration is required thus bringing more investment from both parties which might not be always an ideal approach in the projects where clients are mostly in hurry, but yes initially propose a fixed amount on a set of stories & keep I&A
-> Keep everything flexible but with a set of scope, schedule and price.

Price is tricky, and Fixed Price in Agile: It all depends on how well we manage client expectations. Agile recommends : "Fixed Price, Fixed Schedule and Flexible Scope"
But the problem with this approach when it comes to sprints execution is that client started demanding many functionality or features (let's call CRs) which were not truly captured in the scope, hence it is a responsibility of the entire team to keep the continuous communication & collaboration in place with each sprint. **Show less**

Like | 1 ··· 2w

Paul Oldfield @Mahendra - "where continuous collaboration is required" - If they aren't sufficiently engaged to collaborate to get a good outcome, one needs to question whether they are sufficiently committed to the outcome at all.

"the problem with this approach - client started demanding - not truly captured in the scope" - Yes, but that's not a problem if you truly have *flexible scope*. Allow it to flex; give the customer visibility of what would be going out of scope owing to fixed price, if they put new things in scope. And let *them* make the decision whether the swap (or risk of a swap, the bottom of 'achievable scope' is a range not a line) is worthwhile. **Show less**

Like | 11 ··· 2w

Roy Morien The question as asked, and the big question anyway, is about a fixed contract up-front. While not being a good idea from the agile point of view, it need not disturb the agile development activity. What is required does not dictate how it is to be developed. Rapid development is promised by 'agile', so the requirements can be delivered faster. There are two sides to a contract. If the client insists on a fixed contract, then the developer should demand a level of detail than can be truly seen as full requirements, to be specified by the client, and the system is delivered to meet those detailed requirements. No more nonsense from the client about 'Oh, but you should have understood what we really meant" sort of thing. Changes should, of course, be anticipated in the contract with specific agreement on how to handle changes, and what a change is defined as. A 'not in the contract, not in the system' attitude should be adopted, diplomatically of course. **Show less**

Like | 1 ··· 1w

#12.

Fabian Pachner ··· 1w
Senior Software Developer at TeamViewer

Daily Scrum: Should the ScrumMaster be there?

According to the Scrum guide (and several Scrum assessments), the daily Scrum is only obligatory for members of the development team. As a key task of a ScrumMaster is to remove impediments from the team members, I would think that it is a wasted opportunity for the ScrumMaster not attending the daily. The same might hold for the PO. How do you handle this in your company/team and what are your experiences? **Show less**

Like Comment | 23 95

#Selected comments

Tom Mellor Niels, having worked directly with Ken Schwaber for a few years, the Scrum Guide is a.....guide! And 8 years as a CST (by Ken) and with over 3500 CSM/CSPO attendees, I can say with some certainty that I am familiar with and teach to the Guide with caveat that it is a.....guide! Years back when Ken was a mentor to me, I discussed with him about people serving in the role of Scrum Masters also working on the Scrum Team. His response: Why not? So long as services provided by the Scrum Master comes first. I assure you we were not doing Scrumbut; it was Scrumand. And, we were agile.

Lisa, I can also attest from being mentored directly by Ken that the responsibilities of a Scrum Master do not include keeping a team "moving forward and happy. " The responsibilities do include removing impediments, assuring everyone understands and uses Scrum effectively, and providing service to Product Owners, Dev Teams and the organization as described on pages 6 and 7 of the Guide. **Show less**

Like | 2 ··· 2d

Paul Oldfield The meeting is a meeting for the team. Normally I would expect the SM to attend and observe, but I wouldn't make attendance compulsory for the SM. If, for example, there is opportunity to make good progress with a major impediment but only by missing the Daily Scrum, then I would not be surprised if the removing of the impediment took precedence.

PO? It depends. I have a lot of sympathy with @Allen's point of view, but if other channels of communication between Team and PO are working well, then maybe the PO doesn't need to attend. What works well in *your* context? **Show less**

Like | 5 ··· 2w

Padma Satyamurthy While agree to Allen that its good for SM to attend the daily stand ups, I would still recommend to make it to suit the context. A matured team that is self organizing is quite capable of leading the meeting without a SM. SM and PO can be brought in as required by the team.

Like | 3 ··· 2w

#13.

Paul Grew
Agile Coach

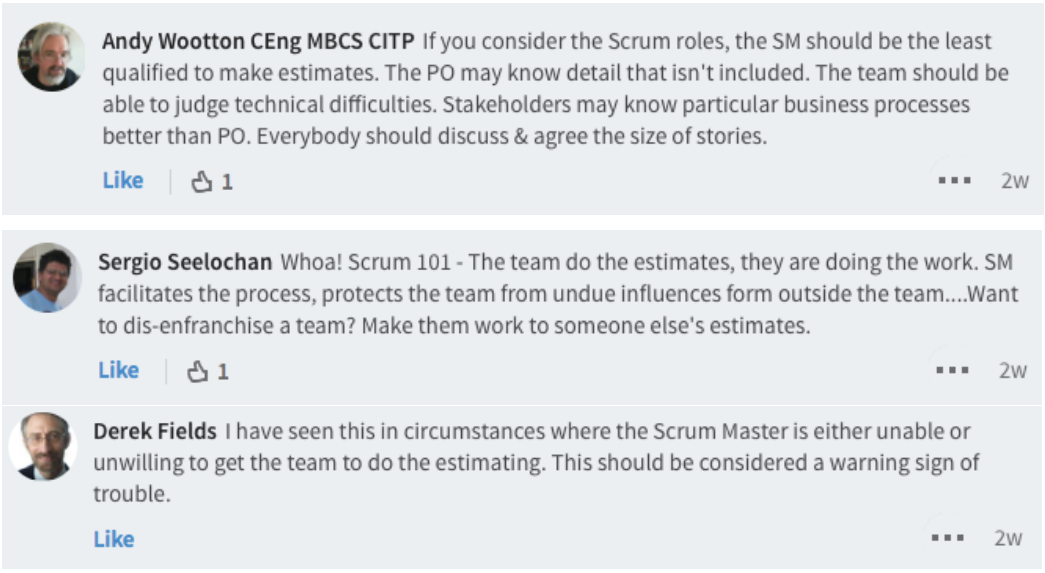
··· 2w

scrum master estimating

I've seen a few occasions recently where the scrum master estimates the backlog. anyone else seeing this?

Like Comment | 6 40

#Selected comments



Andy Wootton CEng MBCS CITP If you consider the Scrum roles, the SM should be the least qualified to make estimates. The PO may know detail that isn't included. The team should be able to judge technical difficulties. Stakeholders may know particular business processes better than PO. Everybody should discuss & agree the size of stories.

Like | 1 ··· 2w

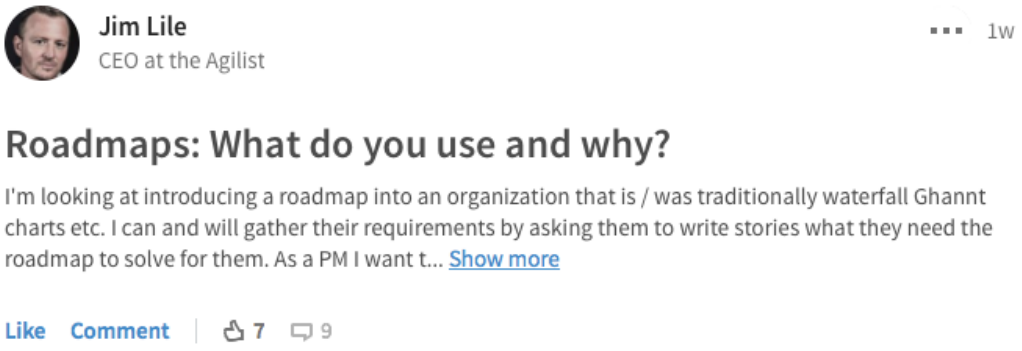
Sergio Seelochan Whoa! Scrum 101 - The team do the estimates, they are doing the work. SM facilitates the process, protects the team from undue influences form outside the team....Want to dis-enfranchise a team? Make them work to someone else's estimates.

Like | 1 ··· 2w

Derek Fields I have seen this in circumstances where the Scrum Master is either unable or unwilling to get the team to do the estimating. This should be considered a warning sign of trouble.

Like ··· 2w

#14.



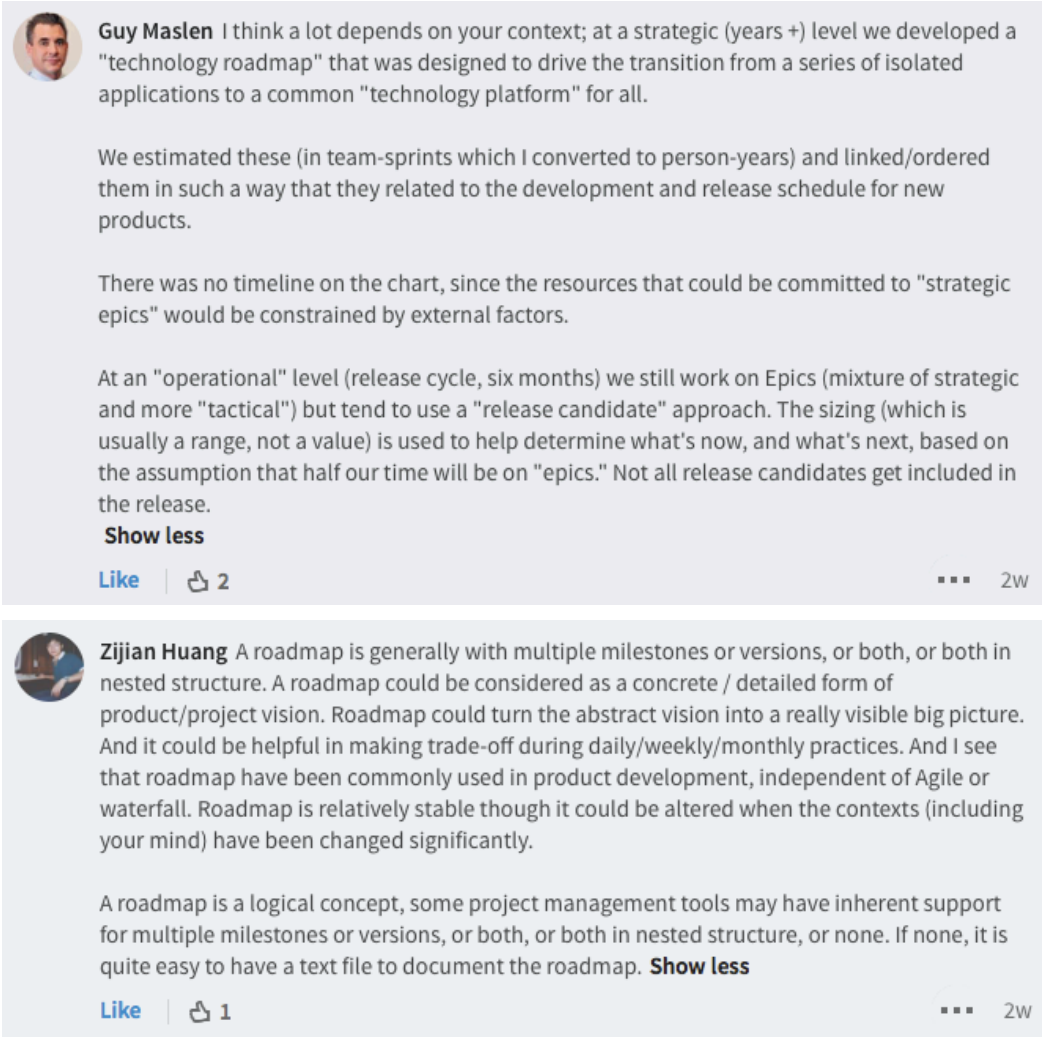
Jim Lile
CEO at the Agilist

··· 1w

Roadmaps: What do you use and why?

I'm looking at introducing a roadmap into an organization that is / was traditionally waterfall Ghannt charts etc. I can and will gather their requirements by asking them to write stories what they need the roadmap to solve for them. As a PM I want t... Show more

Like Comment | 7 9



Guy Maslen I think a lot depends on your context; at a strategic (years +) level we developed a "technology roadmap" that was designed to drive the transition from a series of isolated applications to a common "technology platform" for all.

We estimated these (in team-sprints which I converted to person-years) and linked/ordered them in such a way that they related to the development and release schedule for new products.

There was no timeline on the chart, since the resources that could be committed to "strategic epics" would be constrained by external factors.

At an "operational" level (release cycle, six months) we still work on Epics (mixture of strategic and more "tactical") but tend to use a "release candidate" approach. The sizing (which is usually a range, not a value) is used to help determine what's now, and what's next, based on the assumption that half our time will be on "epics." Not all release candidates get included in the release.

Show less

Like | 2 ··· 2w

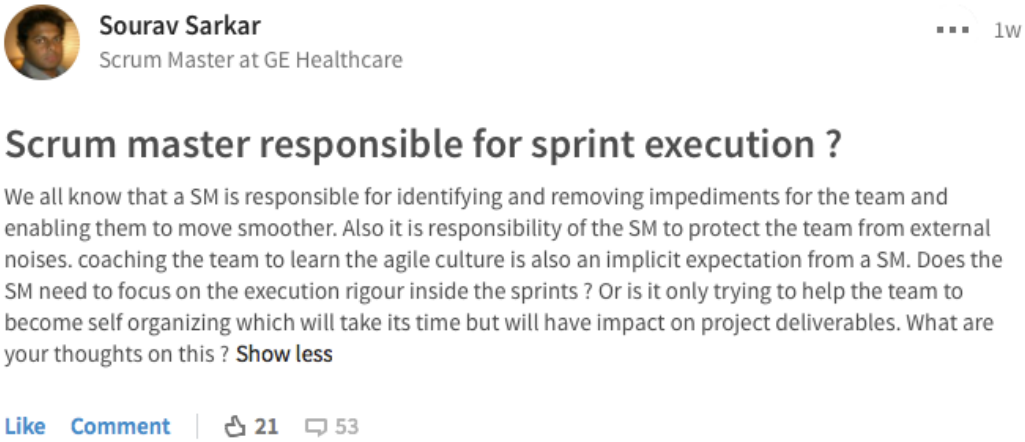
Zijian Huang A roadmap is generally with multiple milestones or versions, or both, or both in nested structure. A roadmap could be considered as a concrete / detailed form of product/project vision. Roadmap could turn the abstract vision into a really visible big picture. And it could be helpful in making trade-off during daily/weekly/monthly practices. And I see that roadmap have been commonly used in product development, independent of Agile or waterfall. Roadmap is relatively stable though it could be altered when the contexts (including your mind) have been changed significantly.

A roadmap is a logical concept, some project management tools may have inherent support for multiple milestones or versions, or both, or both in nested structure, or none. If none, it is quite easy to have a text file to document the roadmap. **Show less**

Like | 1 ··· 2w

#Selected comments

#15.



Sourav Sarkar
Scrum Master at GE Healthcare

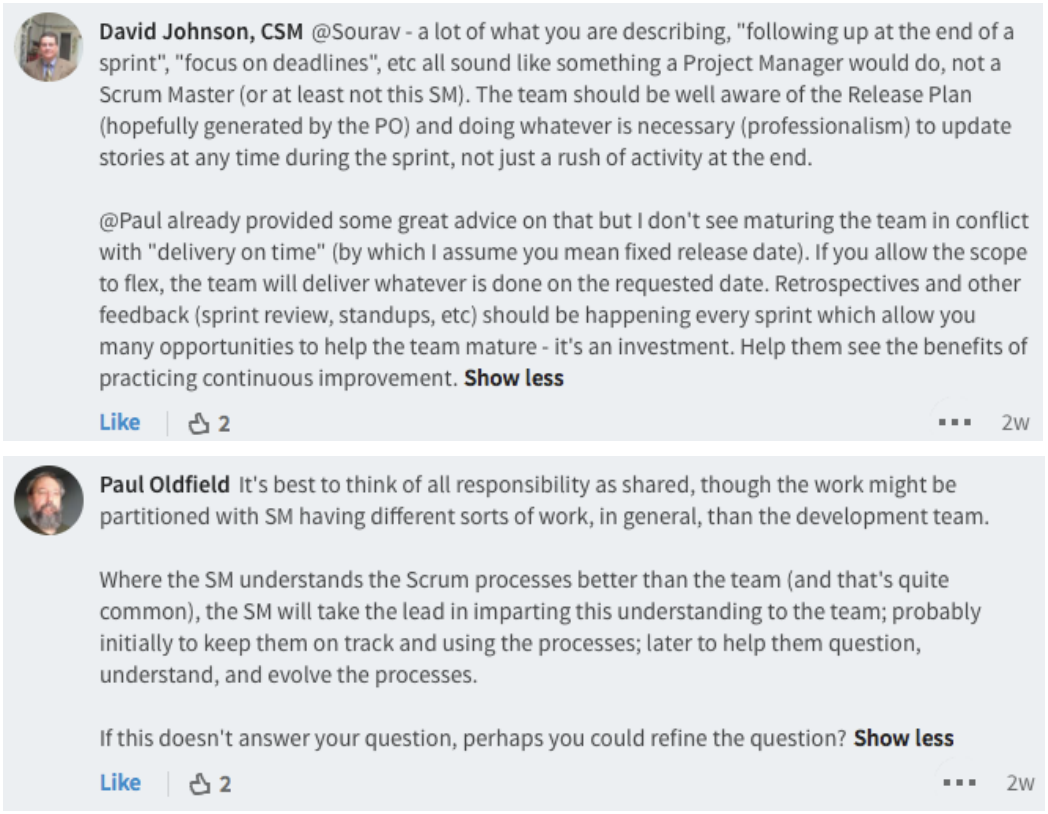
··· 1w

Scrum master responsible for sprint execution ?

We all know that a SM is responsible for identifying and removing impediments for the team and enabling them to move smoother. Also it is responsibility of the SM to protect the team from external noises. coaching the team to learn the agile culture is also an implicit expectation from a SM. Does the SM need to focus on the execution rigour inside the sprints ? Or is it only trying to help the team to become self organizing which will take its time but will have impact on project deliverables. What are your thoughts on this ? **Show less**

Like Comment | 21 53

#Selected comments



David Johnson, CSM @Sourav - a lot of what you are describing, "following up at the end of a sprint", "focus on deadlines", etc all sound like something a Project Manager would do, not a Scrum Master (or at least not this SM). The team should be well aware of the Release Plan (hopefully generated by the PO) and doing whatever is necessary (professionalism) to update stories at any time during the sprint, not just a rush of activity at the end.

@Paul already provided some great advice on that but I don't see maturing the team in conflict with "delivery on time" (by which I assume you mean fixed release date). If you allow the scope to flex, the team will deliver whatever is done on the requested date. Retrospectives and other feedback (sprint review, standups, etc) should be happening every sprint which allow you many opportunities to help the team mature - it's an investment. Help them see the benefits of practicing continuous improvement. **Show less**

Like | 2 ··· 2w

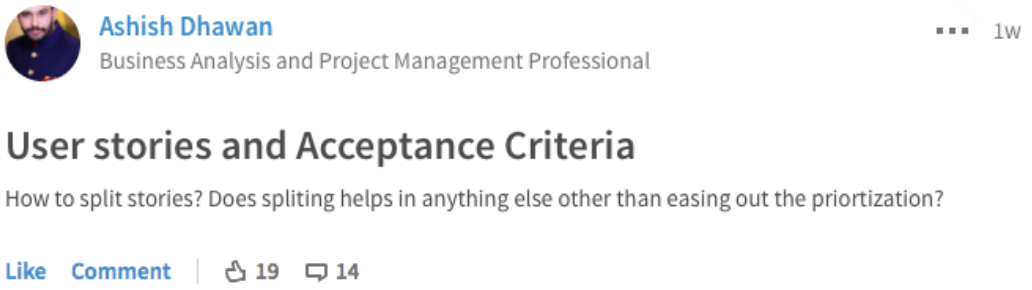
Paul Oldfield It's best to think of all responsibility as shared, though the work might be partitioned with SM having different sorts of work, in general, than the development team.

Where the SM understands the Scrum processes better than the team (and that's quite common), the SM will take the lead in imparting this understanding to the team; probably initially to keep them on track and using the processes; later to help them question, understand, and evolve the processes.

If this doesn't answer your question, perhaps you could refine the question? **Show less**

Like | 2 ··· 2w

#16.



Ashish Dhawan ··· 1w
Business Analysis and Project Management Professional

User stories and Acceptance Criteria

How to split stories? Does spliting helps in anything else other than easing out the priortization?

Like Comment | 19 14

#Selected comments

Christopher Bimson Try splitting a story up based on the acceptance criteria. You'd be surprised how often it's possible to easily frame the acceptance criteria for a larger story as stories in their own right.

You can also try the cheat sheet at: http://agileforall.com/resources/how-to-split-a-user-story/

Splitting helps planning and (if you do it) estimation.

You should also find it helps you deliver earlier with less effort. The bigger a story is, the more likely it is to have 'fat' that can be trimmed easily when the story is split. **Show less**

Like | 4 ··· 2w

Allen Holub At the macro level, a story that takes too long to implement will make your feedback loop too large. You never want to go larger than 2 weeks or so, but even that's too large. For one thing, there's no way to estimate implementation time with any accuracy, so if the think the story will take 2 weeks, odds are it will take longer. The other problem is that you need to be able to compute an accurate average velocity (measured in stories-per-week) to make necessary business projections. To do that, the stories have to be relatively small (a few days at most). **Show less**

Like | 4 ··· 2w

Paul Eastabrook ... (Continued) beyond INVEST however, we often deal with different size stories over time. For example, "Epic" stories whilst useful for high level release planning, are useless and not sufficient for iteration/sprint planning and development purposes.

Breaking down stories from BIG rocks into smaller rocks as they get closer to the time they will make it into an iteration/sprint/flow is essential for reducing waste and emerging detail just in time, whilst maintain a sense of the bigger picture.

I enjoy the story mapping technique for the purpose of relating stories to the overall vision and having a sense of emerging detail as the stories get broken down and split closer to the time of being worked upon by a cross functional team. **Show less**

Like | 1 ··· 2w

Paul Oldfield Having small stories certainly helps. If you can get the stories to a consistent small size you can even dispense with Story Points, replacing it with a count of User Stories.

So, smaller stories mean you can defer the less valuable bits, you can have a nice, relatively even cadence on stories themselves, you can deliver sooner (if doing continuous delivery), and you can dispense with estimating in Story Points. You might think this is enough to make you want to learn how. It probably is. And eventually it becomes easy.
Show less

Like | 1 ··· 2w

#17.

Lisa Brice, PMP
Sr. Project Manager at Trupanion - Medical insurance for your pet

1w

Calculating Sprint Performance (when everything went to hell)

Hello, I have a complicated question. Well, at least I've been having issue with it. Our team works in sprints. We recently completed what was scheduled to be a two-week sprint (with one week release cycle) that included 80 story points. However, due... Show more

Like Comment | 14 55

#Selected comments

Chris Alexander, SCT, SMC, SAMC, ACP, HPT Hi Lisa - my first observation is that there seems to be some fundamental misunderstandings on what a Sprint is. A Sprint is a timebox, and is fixed. Hence, you cannot have a 15 day Sprint which takes 34 days. A 15 day Sprint would take - 15 days. Incomplete work is then carried over into the next Sprint (which shouldn't be happening ideally, but occasionally does). Also, Sprints are not coupled to releases (in principle). The development and release of features are independent of one another, at least at planning and conceptual levels. (Which isn't to say you can't release a feature at the end of every Sprint - certainly you can, but that is a purposeful decision, not the point/focus of the process itself.) Please feel free to reach out if you'd like to discuss more in-depth. **Show less**

Like | 1 2w

Brett Maytom I assume you are using Scrum from the terminology in your post.

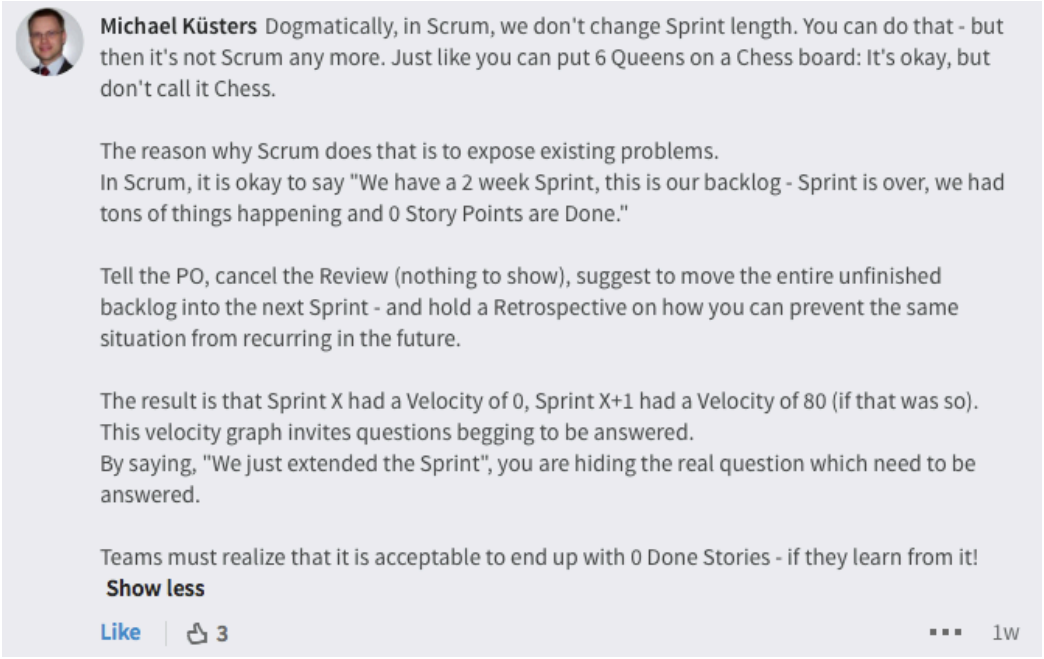
A sprint is a fixed time box and at the end of the time box the sprint is over and you go into review retrospective. It appears you chose to extend the sprint to 34 days which totally defeats the point of forced inspection.

You missed the critical feedback and inspection cycle of the events to manage expectations, really understand what was going on and then to create a new plan. Instead the team pushed forward with no empirical processes. This is traditional project thinking what you need to stop.

At the end of the sprint, it is over. You actually ran four sprints. The first 3 were missed opportunities to adapt and learn.

My recommendation is stop this practice and each sprint you force inspection. Talk about the challenges and adapt. This is the point of sprints. **Show less**

Like | 1 2w



Michael Küsters Dogmatically, in Scrum, we don't change Sprint length. You can do that - but then it's not Scrum any more. Just like you can put 6 Queens on a Chess board: It's okay, but don't call it Chess.

The reason why Scrum does that is to expose existing problems.
In Scrum, it is okay to say "We have a 2 week Sprint, this is our backlog - Sprint is over, we had tons of things happening and 0 Story Points are Done."

Tell the PO, cancel the Review (nothing to show), suggest to move the entire unfinished backlog into the next Sprint - and hold a Retrospective on how you can prevent the same situation from recurring in the future.

The result is that Sprint X had a Velocity of 0, Sprint X+1 had a Velocity of 80 (if that was so).
This velocity graph invites questions begging to be answered.
By saying, "We just extended the Sprint", you are hiding the real question which need to be answered.

Teams must realize that it is acceptable to end up with 0 Done Stories - if they learn from it!
Show less

Like | 3 ··· 1w

Marjorie Pries Keeping a fixed sprint length is useful in that it helps you figure out what the team can really do in that amount of time. Did any stories get finished and pass QA in your two week frame? Those that did result in the sprint velocity and that lets the team plan what is realistic for the next fixed-length sprint.

If you frequently have sprints of 0 points completed, it's time to retro over Why. Maybe you need longer sprints but maybe you need to re-think your stories. Maybe the team is putting on a facade of Agile when it is still being asked to develop everything all at once and then test everything all at once in a waterfall way... **Show less**

Like ··· 1w

#18.

Gerald O'Connor ··· 1w
Software Development Manager (Agile Project Manager) at Relay Wealt…

How to handle user stories that are waiting for something?

Some user stories are waiting for responses from customers. Some are waiting on domain experts. Some are waiting for the product owners input.

It's not all of the user stories all the time, but some of the user stories all the time.

It's a nuisance... **Show more**

Like Comment | 1 14

#Selected comments

Michael Küsters Try making a simple model and visualize *where* the user stories are stuck, *how long* they are stuck and ask in the team/organization: "While our competitors simply *do* things and conquer the market, we sit here and delay value. What can we do about it?"

A "Waiting user story" is either irrelevant (i.e. the wait does not matter) or it's Value Denied (i.e. we lose money with every day that the story is not Done).

Simply delete all of the first stories from the backlog, since they're obviously waste.
Attach a loss value to those that have an opportunity cost for waiting and look first at those with the highest cost.
Waiting for input is not a god-given law of the universe - it's a choice that someone made. Make it the problem of someone who can solve this and educate them to choose more wisely.

What is the reason why people wait for input rather than work with assumptions and learn from feedback? **Show less**

Like | 2 ▪▪▪ 1w

Broughan Macklin Put a big red ticket on your Scrum board to highlight the impediments for each story, anyone who walks past your team's board will see the stories are blocked and hopefully start a conversation as to why. Or, at the end of the sprint when you have a bunch of stories still WIP, bring in all the red tickets you had throughout the sprint and start a discussion on what the team can do differently. **Show less**

Like | 1 ▪▪▪ 1w

Paul Eastabrook Straight off the bat visualise it some how. If it is one the story wall or Kanban make sure they are modelled in the appropriate process stage and BLOCK them with the next action/waiting for.

Too really put the cat amongst the pigeons, WIP limit the process stages. This will force the issue of collaboration and unblocking those items, plus highlight potentially scary levels of work in process you have and queue sizes.

Ultimately it sounds like you are accruing wasteful inventory potentially unnecessarily and don't have the right people do using on the right work to flow more stuff to done over any given period of time. **Show less**

Like | 1 ▪▪▪ 1w

#19.

Anmol Tuteja
Consultant at ARM

5d

What works best in Service operations which keeps getting lots of incidents and requests? Kanban?Or a one week sprint?

#agile #scrum #Kanban #scrumban

Like Comment | 11 18

#Selected comments

Raazi Konkader Anmol, the devil is in the detail.
Understand your business context first...
1) Incoming demand
2) Current throughput ... Or rather your capability to fulfill that demand

Plot both against the "need"...SLAs / OLAs

Once you have demand data:
Understand the incoming rate
Understand trends
Understand clusters
The dimensions to plot this against: severity, priority, value impact, reputational impact.

Plot this against your current throughput or rather your strategy that delivers that throughput.

Now, see the gap that remains with the stated "need".... I.e. Your SLA

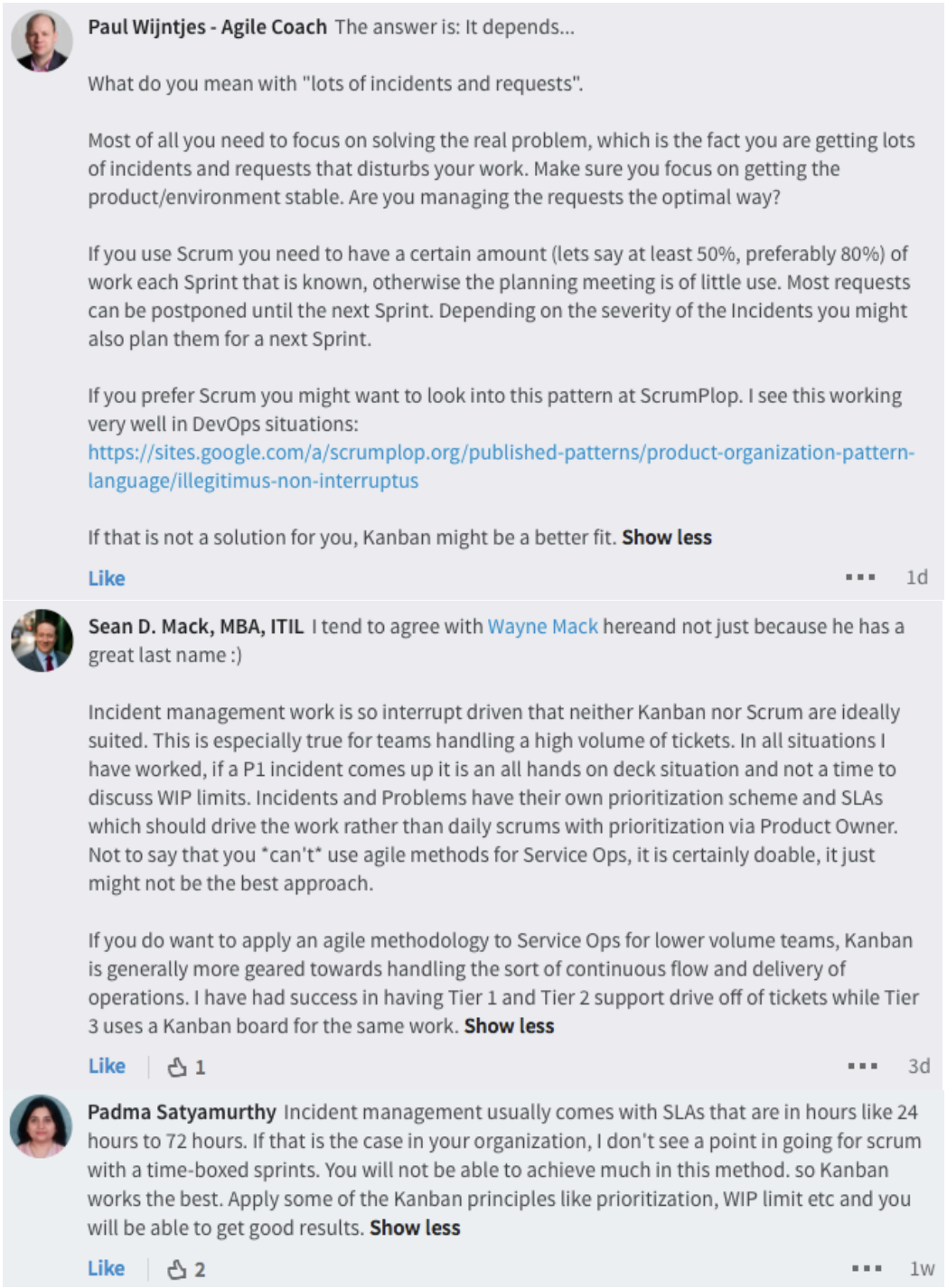
This gives you your business context.

The choice of Kanban (iterationless flow based systems) vs Scrum (batch based ceremony driven framework) is then a function of the gap in your business context... Don't discount the fact that you could have your team operating at two cadences .., one in Kanban mode for the unpredictable demand n one in Scrum for the predictable demand. **Show less**

Like 1w

David Denham If you're using Scrum for this, what would be the point in having a Sprint Review, for example? You are not really looking to learn from feedback and iterate over the product. You're taking requests and bugs in where their outcome is known. I would describe these as complicated work, rather than complex. Kanban handles complicated work well - so would probably be a better fit. **Show less**

Like | 1 1w



Paul Wijntjes - Agile Coach The answer is: It depends...

What do you mean with "lots of incidents and requests".

Most of all you need to focus on solving the real problem, which is the fact you are getting lots of incidents and requests that disturbs your work. Make sure you focus on getting the product/environment stable. Are you managing the requests the optimal way?

If you use Scrum you need to have a certain amount (lets say at least 50%, preferably 80%) of work each Sprint that is known, otherwise the planning meeting is of little use. Most requests can be postponed until the next Sprint. Depending on the severity of the Incidents you might also plan them for a next Sprint.

If you prefer Scrum you might want to look into this pattern at ScrumPlop. I see this working very well in DevOps situations:
https://sites.google.com/a/scrumplop.org/published-patterns/product-organization-pattern-language/illegitimus-non-interruptus

If that is not a solution for you, Kanban might be a better fit. **Show less**

Like ▪▪▪ 1d

Sean D. Mack, MBA, ITIL I tend to agree with Wayne Mack hereand not just because he has a great last name :)

Incident management work is so interrupt driven that neither Kanban nor Scrum are ideally suited. This is especially true for teams handling a high volume of tickets. In all situations I have worked, if a P1 incident comes up it is an all hands on deck situation and not a time to discuss WIP limits. Incidents and Problems have their own prioritization scheme and SLAs which should drive the work rather than daily scrums with prioritization via Product Owner. Not to say that you *can't* use agile methods for Service Ops, it is certainly doable, it just might not be the best approach.

If you do want to apply an agile methodology to Service Ops for lower volume teams, Kanban is generally more geared towards handling the sort of continuous flow and delivery of operations. I have had success in having Tier 1 and Tier 2 support drive off of tickets while Tier 3 uses a Kanban board for the same work. **Show less**

Like | 1 ▪▪▪ 3d

Padma Satyamurthy Incident management usually comes with SLAs that are in hours like 24 hours to 72 hours. If that is the case in your organization, I don't see a point in going for scrum with a time-boxed sprints. You will not be able to achieve much in this method. so Kanban works the best. Apply some of the Kanban principles like prioritization, WIP limit etc and you will be able to get good results. **Show less**

Like | 2 ▪▪▪ 1w

#20.

Tiffany Vockins
Software tester at CharityJob

··· 3d

Agile Testing

What do testers do during the first couple of days of a sprint when no development has been completed?

Like Comment | 22 40

#Selected comments

Allen Holub @Tiffany, They build the acceptance tests, which should exist before you start writing the code itself. There are no passive testers (people who just bang on keys with no understanding of how to write code) on the team. That sort of testing (often pejoratively called "monkey testing") has no place in an Agile team---you just don't have time for it. Tests must be automated.

Moreover, everybody should be a "generalizing specialist," and in the case of a tester, they need to generalize into coding, at least at the level where they can put together an automated acceptance test. If your "testers" don't know how to code, teach them.

I should also say that I'm unhappy with the notion of dedicated testers on an agile team (or dedicated anybody for that matter). Seems to me that you've just set up a mini waterfall within the team. You need more parallelism than that. Testing should be going on constantly, not in a after-the-code-is-done waterfall phase. **Show less**

Like | 3 ··· 1w

Janet Gregory Gus said it very well in his blog post, although one thing I didn't see in Gus's post was the benefit of exploratory testing. Automation can only check what you thought about ... Exploratory testing (not monkey testing) has tremendous value looking for realistic use of the product, and things we did not think about.

Having a dedicated tester on a team does not mean mini-waterfall. That only happens if they are not a true collaborative team member or if the team does not take ownership of quality. **Show less**

Like | 5 ··· 1w

Carla Severi The work of an Agile Tester is not only to write Tests.
First of all, the agile tester haves a lot of conversations with other members of the team during all the Sprint Session.
He also speaks to and question the PO to understand perfectly real value to be delivered is and he contributes to build a better product.
He finally writes of the scenarios and he decides with the PO which regression tests have to be prioritized.
It's a job that request a lot of effort.
An agile tester has no free time during his working day, donot' worry :-) **Show less**

Like | 4 ··· 1w

CBS
COPENHAGEN
BUSINESS SCHOOL
HANDELSHØJSKOLEN