Retrospective Process Tracing

A Waste Identification Technique in Lean and Agile Software Development

Kristoffer Aalhus

Master's Thesis in Information Science

University of Bergen

29.05.17

2017

Retrospective Process Tracing

Kristoffer Aalhus

https://bora.uib.no/

Abstract

This master's thesis describes a new waste identification technique, adapted from the Process Tracing sub-method *Explaining Outcome* described in Beach et al. (2013). The aim of the technique is to enable development teams, that adhere to the principles of Agile and Lean development, to identify waste that is the result of the teams' uniqueness in team and organizational structure, which also cannot be directly identified by looking at the literature of *waste categories* or Agile *success factors*. The thesis cites the findings of a three-month long exploratory Action Research study, involving six team members developing a travel expense reimbursement system. The technique was found to enable a cost-efficient waste identification process within the team, and could serve as a substitute for the traditional Retrospective Meetings as described in the Scrum Guide. The findings are believed to be of importance to teams of similar parameters, but verification studies should be held to better understand the technique's scalability and generalizability. The thesis should be of interest to practitioners and academics alike, and especially so for project managers of small-scale development teams.

Acknowledgements

There are certain people whom deserves my sincerest gratitude, and have supported me in varying capacities while writing this master's thesis: These include my supervisor and associate professor Solveig Bjørnestad for much needed help and advice; Jørund B. Vedøy for providing unspecified amounts of coffee; Simen Skaret Karlsen for proof-reading; and not least Inni Mowinckel for coping with my growing insanity. In addition, I would also like to thank the participants for letting me study the team during the Action Research phase; professor Bjørnar Tessem for hosting the Process Tracing sessions; and Simen Jensen for our additional discussions about the method.

Table of Contents

Table of Figures

1. Introduction

Agile and Lean software development have throughout the last fifteen years gained a solid footing as the framing methodologies for working in rapidly changing markets, handling customer-side requirement-insecurities, and minimizing the impact of documentation on an organization's bottom-line. Dethroning traditional development techniques and other organizational structures by promoting value-creation over top-down control, both Agile and Lean methodologies are, by the time of writing, the *de facto* solutions for large and small software development projects alike.

The success and popularity of both Agile and Lean methods can be attributed to several reasons. The most commonly cited success factors of Agile and Lean development include a closer relationship with the customer (*both up-front commitment and collaboration*), revived corporate culture (*flat organization, collective code-ownership*) and a renewed sense of control (*short, but regularly held meetings, in addition to reviews and demonstrations with the customer*) (Misra et al., 2015). It is arguably the sum of all these aspects that make Agile and Lean strategies work as well as they do in many, diverging settings: From small, in-house projects, to large, distributed commercial undertakings.

The success factors cited above all contribute to make development effective and efficient, in that it ensures that the developers are producing the right features, at the right time, while promoting self-organization. However, this is not a trivial task, and there are many pitfalls that can hinder development pace if not addressed accordingly. Development teams try to alleviate such pitfalls by promoting self-evaluation following a Sprint, usually performed by holding *Retrospective Meetings*.

Events, procedures and processes that negatively influence Development Velocity, collectively known as *waste,* is a subject of on-going study in the field of software engineering and development methodology research. However, much of the research into process optimization, or the identification of waste, has been put forward under the banner of *critical success factors*, or the factors that are necessary in securing rapid development. In addition, some research has been concerned with *waste heuristics*, meaning collections of tell-tales that indicate that some event, process or procedure within a development team is suffering from (empirically) grounded factors that have been shown to slow Development Velocity (*for instance, Ikonen et al., 2010 or Womack et al., 1996)*.

Both research into critical success factors and waste heuristics follow the same paradigm: They aim at uncovering generalizable facts about waste in software development. They do not cope with the reality that development teams vary greatly in parameters, i.e. team size, distribution, or level of top-down management, and that wasteful behaviour in one team might not be wasteful in another. *Case-specific* inquiries, or research considering the parameters of a *single team*, have been historically circumscribed for the advent of generalizability. As such, there is close to no research that can help project managers, Scrum Masters and so forth, in providing tools that can identify the specific problems that their own team might face, with exception of heuristics and categorical answers; often leaving them guessing.

This concern has motivated me to study this problem. As a struggling project manager of a start-up company, research into waste was the first and foremost interesting field of study that I thought needed additional organizational tools, as the regular Retrospective Meetings, and other techniques of Agile and Lean development, simply were not sufficient in establishing an effective, self-organising team structure and organizational parameters. The literature I found was not specific enough to be of sufficient aid in solving on-going problems.

In trying to alleviate this problem, I decided to research the family of methods called Process Tracing (PT) as a possible waste identification tool, coping with the unique organizational parameters of a single team. PT has previously been used successfully in social science, in mapping out causal chains of events in organizational constructs, and hopefully this feature will show to hold true also in the identification of waste, in the field of software development. However, it has been used to very little extent in Information Systems and Software Engineering research, and how well this analytical framework performs with this novel usage will therefore be the subject of this thesis.

Process Tracing does seem, however, to be a good fit for such a case-specific inquiry. Traditionally, it has been applied to political science, among other fields, in trying to understand causal mechanisms and decision-making. Wasteful behaviour in development teams might also be possible to understand from a temporal and causal point of view, and therefore seems as a viable starting point for such an inquiry. The merits of Process Tracing will be detailed in the coming chapters.

A technique that can enable development teams to identify what is wasteful in their development and organizational structure can hold unique value for both the team *and* the organization it operates within, both in terms of financial and human resources. Therefore, the main motivation behind this thesis is to provide those interested in improving their own development teams' processes an organizational tool that is scalable and easy to implement in your own, unique setting. As a tool, it must cope with the parameters of specific teams, such as size and organizational situation, and help making the self-improvement routine both more efficient and self-driven, as well as enabling the users to unearth waste-objects directly related to their team's uniqueness in organizational parameters.

When researching Retrospective Process Tracing, I decided to apply the method named Action Research. This was done to enable the participants of the study to share their experiences along the way, and to collectively build a Retrospective tool that was suited to all members and situations of the development team.

To establish the usability and efficiency of Process Tracing as a supplement or alternative to Retrospective Meetings for identifying waste, the research aimed at answering the following three research questions:

Research Question I: *Is it possible to identify waste-objects not covered in the existing waste literature by using Retrospective Process Tracing?*

Hopefully, using Process Tracing as a Retrospective tool will enable development teams to identify sources of waste that are not previously described in the literature, i.e. single-case, single-organization and specific waste-objects which are the result of unique patterns, procedures and work habits of the team utilizing the method. If the research can showcase a single-case waste-object not covered by the literature, we have evidence for the viability of such an organizational tool.

Research Question II: *Can Retrospective Process Tracing be implemented with a relatively low initial cost?*

There are many sources of waste. If the utilization of Process Tracing as a Retrospective tool requires much resources and does not significantly increase development efficiency or decrease waste, it itself is a source of waste, and should be discarded as a viable technique for all practical purposes.

Research Question III: *Is Process Tracing enough to serve as a substitute to traditional Retrospective Meetings, or should it be used as a supplement?*

If there are any aspects of traditional Retrospective Meetings that Process Tracing fails to address sufficiently, then those aspects should be transferred from the traditional self-inspection technique to this new one.

2. Background

To provide a foundation for the forthcoming discussions and argumentation, there are primarily four segments that are worth investigating in greater detail. These are *Agile & Lean principles and implementations, waste*, traditional *identification techniques* that have been formerly used, as well the distinct types of *Process Tracing* the literature has described. This chapter will be structured into four parts, each highlighting one of these segments. This should provide the reader with a sufficient understanding of all the related terms, definitions and concepts needed to follow the thesis.

2.1. Agile & Lean – Principles and implementations

Agile and Lean denotes two, partly disjoint, schools of methodology. Specifically, they constitute sets of principles for development and manufacturing that form the basis upon which process frameworks, such as Scrum, are built. Even though they share some similarities, they are sufficiently different to warrant a closer look. This section will highlight the principles of both Agile and Lean, as well as their two most popular implementations in the field of software development, at the time of writing: Scrum and Kanban.

2.1.1. Lean principles

When considering Lean development, Poppendiek et al. (2003) denotes seven underlying principles worthy of discussion, and by adhering to these, one should be able to classify oneself as a Lean development team. These are discussed beneath.

2.1.1.1. Eliminate waste

Poppendiek et al. (2003) defines waste as "*anything that does not add value to a product,* [...] *as perceived by the customer"*. Therefore, understanding what waste is has become a broad topic spanning all parts of system development, from organizational procedures to the everlasting discussion of space vs. tab indentation. Anything goes: If it does not add value for the customer, get rid of it. The ideal is to instantly address the customer's need, ensuring that their satisfaction is met.

2.1.1.2. Amplify learning

Developing software is a complex task, and its complexity multiplies when developing software with others. Therefore, "*the best approach to improving a software development environment is to amplify learning*" (Poppendiek et al., 2003). This might be handled by implementing feedback loops, learning cycles, or by distributing the innate knowledge of team members onto others. Making sure that people grow on the job is important; both for the team and the customer.

2.1.1.3. Decide as late as possible

Because of the volatility of working with external customers, or developing products in uncertain markets, it is better to decide on what to do later, rather than sooner. This enables a development team to base their decisions on *more* information, making for better judgements. However, this principle also entails a need for contingency plans, as well as a capacity for both expecting and responding to changes.

2.1.1.4. Deliver as fast as possible

Poppendiek et al. (2003) states that *"the discovery cycle is critical for learning",* by which they mean that getting feedback is necessary for knowing that one is developing the right thing: And one can only get such an understanding by delivering fast and often. Only with short cycles of employment can one be certain that one is on the right track. This also enables learning; about the market, the customer, and your own team's processes.

2.1.1.5. Empower the team

A fundamental principle in Lean development is that it is those who are developing a system that best knows how to develop it. To produce a quality system, one must include the technicians in planning. Instead of instilling strict top-down management, make sure that the team have everything they need, and trust them to get it done.

2.1.1.6. Build integrity in

Integrity can be specified on two separate levels: Software and conceptual integrity. Conceptual integrity refers to a product's level of usability, whether it is adaptable to future scenarios, and whether it "*works together as a smooth, cohesive whole*" (Poppendiek et al., 2003). This principle touches, in part, onto the field of Interaction Design, and essentially states that measuring customer perception of your product is important. Software integrity, on the other hand, refers to how scalable and adaptable the code base is for new scenarios, as well as how maintainable it is over time.

2.1.1.7. See the whole

Avoiding sub-optimization is important. By this, Lean considers there to be more value in understanding the bigger picture of a system development process, than there is in understanding its minute details. Measuring individual performance over collective performance is, by Lean, understood as a poor solution. When developers "watch their own back" rather than considering the collective good, the project will suffer.

2.1.2. Agile principles

The primary source on Agile principles is that of the Agile Manifesto (Beck et al., 2001), which mentions twelve principles denoting the Agile way. Some of these principles are, however, nearly interchangeable with that of Lean. For instance, when Lean considers eliminating waste, Agile answers by stating that its highest priority is to satisfy the customer; when Lean considers deciding as late as possible, Agile answer by stating that one should welcome changing requirements. Therefore, many of the principles are so equal in character that it makes little sense simply restating them. Instead, let's consider the differences and additions that Agile imparts.

2.1.2.1. Reflection

A core idea behind Agile is to promote reflection and self-inspection. To grow as a team, it is important to decode one's own working habits, processes and procedures, making sure that the team learn from past mistakes. Enabling a team to reflect on their own work will ensure more effective and efficient collaboration in the future; both internally and externally.

2.1.2.2. Simplicity

"The art of maximizing the amount of work not done" (Beck et al., 2001) is another core idea that forms the foundation of Agile. Don't code for the future, because it will often change. Ensure that you create the minimal amount of code to serve the specific task that you want to solve, and handle new situations as they emerge. However, this does not imply that one should be naïve, but only that developing additional functionality "that we most certainly will need in the future" should not take precedence over the current, agreed-upon, prioritized work.

2.1.2.3. Communication

Team-work relies on communication. In the Agile mindset, face-to-face conversation is always preferable, as it can serve to reduce misunderstandings that might cause problems down the road. Therefore, meeting often and being available for questions is important to ensure the project's success.

2.1.2.4. Cooperation

Software should not be built in silos. Only by cooperation between business people and developers can one expect to develop a system that is aligned with the different stakeholders' needs (Beck et al., 2001). As such, Agile literature often stresses the benefits of cooperation *between* the customer and the team, as well as *within* the team (*for instance, Schwaber et al., 2016)*.

2.1.2.5. Working software as measure for progress

To measure progress within a development team, the first and foremost tell-tale is to look at how much working software the team is able to produce. This forms a base measurement for whether the team is able to improve upon their processes and procedures, from cycle to cycle, during the life-span of a project.

2.1.3. Differences and similarities

At a philosophical level, there are some differences between Agile and Lean. The most important might be stemming from their respective background: Agile, for instance, was created specifically for software development, while Lean has its roots in traditional, industrial production.

Their diverging backgrounds reverberate throughout their respective principles, where, considering the Agile Manifesto, Agile has a greater focus on collaboration between traditional software development stakeholders: The team, the customer and the management. When developing software, there are nearly always multiple stakeholder relations involved. Agile tries to make software development flexible by promoting smaller, shippable increments, which could help the development team easily pivot if or when acquiring new knowledge, or understanding some new perspective of a stakeholder, about the product one is developing.

Additionally, measuring overall progress within a team is a subject where Agile and Lean promotes different values. The Agile Manifesto states that *"working software is the primary measure of progress"* (Beck et al., 2001). The more software a team is able to produce, the better it is. This stance is furthermore highlighted by its focus on how we can define a piece of code, user story or feature to be "done", where acceptance tests[1] are held by the customer to provide a definite "definition of done". Lean, on the other hand, evaluates overall project progress in terms of validated learning (Poppendiek et al., 2003): How sure can we be that this is what the market or the customer wants?

Furthermore, it would also make sense to discuss Lean and Agile *organizations*, and not just development teams. Agile and Lean alike have ramifications spanning further than strictly development processes, where Lean specifically have made a big mark on how to structure an organization (Womack et al., 1996). For instance, new frameworks such as Lean Start-Up have become a hot topic in recent years (Blank, 2013), and concepts such as *value stream* (see section 2.2.3) are still used within the general field of business management.

2.1.4. Implementations

Today, it makes little sense to write about the Lean and Agile mindsets without mentioning the process framework Scrum and the development method Kanban; the most frequently applied applications from the either schools. This section will serve to highlight the important high-level differences between them, but will not serve as a complete description of either.

2.1.4.1. Scrum

The Scrum process framework was invented by Ken Schwaber and Jeff Sutherland, and the Scrum Guide (Schwaber et al., 2016) is the framework's official resource for teams wanting to correctly utilize it. This section will rely on their descriptions when discussing Scrum's varying parts.

[1] Acceptance tests are non-functional tests held by the customer to ensure that their requirements are met.

When working with a project utilizing Scrum, there are primarily three main roles: The Product Owner (one or more representative(s) of the customer whom is ordering the product), the Scrum Master (a facilitator helping to remove any impediments the team might face), and the team responsible for the development. These three roles interface in varying events, and are collectively responsible for ensuring the success of the project.

Scrum works as an iterative process framework for developing software (Schwaber et al., 2016). In Scrum, such iterations are denoted as Sprints, often spanning between two and four weeks. Furthermore, the framework incorporates four high-level items: These are the Product Backlog, the Sprint Planning meeting, the Sprint Backlog, and the Sprint itself.

To understand what to develop, the Product Owner specifies and prioritizes a Product Backlog. This serves as a list of wanted features that the customer would like to have within their system. These are often described as User Stories, taking the following form:

"As a user, I want X such that I am able to Y"

Here, X represents a requirement, while Y represents the end goal that one wishes to complete with that requirement. To exemplify, consider the following: "As a user, I want to view a complete list of time-stamped sales, such that I am able to measure sales performance over time".

This list is solely the Product Owners' responsibility to maintain, but often the development team helps with the prioritization when requirements are interdependent. When having specified such a list, the development team holds a Sprint Planning meeting.

Here, the focus is to evaluate how many items of the prioritized Product Backlog can be developed in the next iteration, presenting a plan to the customer about the overall project expedition. This event is often time-boxed, and aims at establishing a locked-in subset of the Product Backlog that the team commits to develop. When evaluating the amount of work that is possible to complete within the Sprint time-box, the team often utilizes estimation techniques such as *Planning Poker*[2]. The resulting subset deemed possible to complete within the timeframe constitutes the Sprint Backlog, and no additional items are added to this list during the Sprint.

Next up is the actual development cycle; the Sprint. Here, the team organizes between themselves how to develop the Sprint Backlog items, and does so to their agreed-upon standards. During the Sprint, the development team might query the Product Owner about unclear requirements, business-decisions, and so on. However, the team is deemed collectively responsible to produce the functionality that they've committed to, within the time-box that has been agreed-upon.

One way of securing continuous communication between the different team members is to hold Daily Stand-Up Meetings; short sessions, usually time-boxed to no more than fifteen minutes, held at the start of each day, where the goal is to have everybody on the team answering the following questions:

- What have been done since our last meeting?

[2] See Hartman (2009) for additional information about Planning Poker.

- What are you planning to do?
- Do you have any impediments restraining you from following your plan?

If anybody has met such impediments, the Scrum Master tries to alleviate these.

When the Sprint comes to an end, and hopefully the committed-to items have been produced, the development team invites the Product Owner to a Sprint Review. Here, the produced functionality is showcased to the customer, welcoming input and feedback on their work. This meeting serves as a re-alignment of their product vision, and helps the development team in knowing whether they are doing things right; as perceived by the customer. After this meeting, the new features are deployed, the cycle is completed, and the next iteration can start.

This iterative structure grants the development team the necessary information to assess their own work, following the completion of a Sprint. Often, development teams hold Retrospective Meetings at the end of each cycle, where the goal is to inspect what went well, what went wrong, and how one can possibly improve on the latter. This event will be discussed in greater detail in section 2.3.1.

2.1.4.2. Kanban

The Kanban approach is quite different in character, which could be described as a scheduling system aimed at limiting the amount of work-in-progress. This is also visible from its name; Kanban, being a Japanese word, could be translated to English as “queue limiter”. The technique is based upon a pull-system, where functionality is produced one (or at least few) at a time (Gross et al., 2003), enabling the development team to decide how to best implement the requirements set by the customer, based on the maximum amount of information available. The items that are developed are “pulled” from a Product Backlog, ensuring that only a specified number of items are under production at the same time.

An important concept in Kanban is to visualize workflow (Gross et al., 2003). One way of securing the completion of development tasks is to instil a Kanban Board, such as presented in Figure 1.

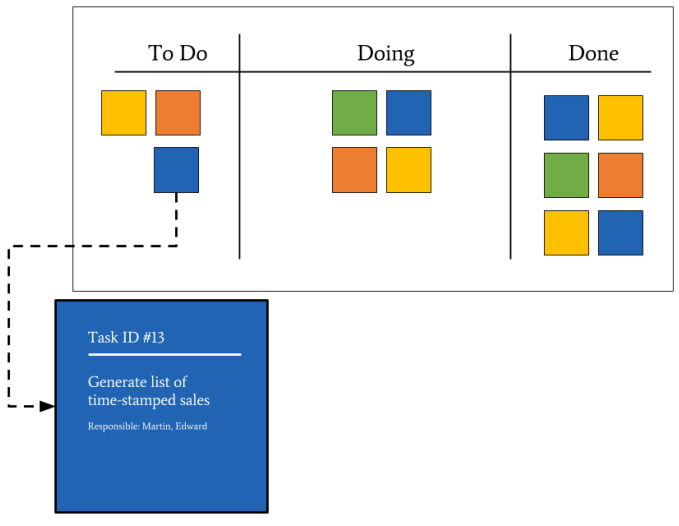

FIGURE 1: GRAPHIC VISUALIZATION OF A KANBAN BOARD.

This board contains at minimum three columns; these are “to do” (i.e. the Product Backlog), the “doing” (items under development), and “done” (finished items where no more work is required). Often, the team might choose to add additional columns, denoting phases such as testing. All the intermediate columns have specified a total allowed number of items, ensuring that features are completed before new ones are started upon.

This way of visualizing work is positive for several reasons. Firstly, the human brain processes visual clues faster than textual (Gutierrez, 2014). Secondly, it enables a quick overview of how much has been done, how much is left, and which of the features are currently under development. Lastly, it enables an easy understanding of which tasks might have dependencies between themselves, and should therefore be developed jointly.

Kanban should to a much smaller extent be considered a *complete* process framework than, for instance, Scrum. Here, the main goal is to describe the processes of the development team, and does not, to an equal extent, provide descriptions of processes involving other stakeholders such as the customer. Therefore, some teams choose to implement this pull-system, which limits work-in-progress, within the framework of Scrum. The mixing of Scrum and Kanban denotes a new type of framework that has been emerging in varying renditions under the collective banner of *Scrumban*. Such variants aim at achieving the best of both Agile and Lean principles, but complicate any concrete description of Scrumban as a concise framework for development.

Kanban differentiates itself further from Scrum in that it is understood as a *continuous* development framework, rather than an *iterative*. Here, there are no cycles involved *(such as the Sprint)*, implying a shorter route from the initial formation of requirements to their subsequent deployment as actual, running code. Lately, new frameworks or concepts such as *DevOps* [3] have sought to harvest this benefit.

2.2. Waste

Both Lean and Agile literature describe and state the importance of continuous process-improvement, and it is this feature, amongst others, that sets these collections of methodology apart from traditional development techniques. Waste, however, has its roots firmly planted in the nascent beginnings of the Lean philosophy. The history of Kanban exemplifies this point, having been responsible for introducing waste as an organizational concept spanning further than any single industry.

2.2.1. History

The Lean way of thinking gained a solid foothold post WWII, where big production companies had to improve their business-procedures to cut cost and gain an edge over competitors. Many view the Toyota Production System (TPS) as a precursor of Lean (for instance; Monden, 2012),

[3] Mueller (2016) provides an in-depth definition of and discussion about DevOps.

and points to the work conducted by their business managers between 1950 up until 1970 as the preliminary foundation of Kanban (Gross et al., 2003). There, the method was originally developed to improve upon the production of physical products. Previously defined processes, such as the inclusion of conveyor-belt production lines championed by Ford Motor Company, were utilized and improved, to eliminate waiting-time and redundant procedures while ensuring that their production was as fast and efficient as possible.

'Just in Time' became an internal slogan at Toyota, meaning that no waiting should occur at any point of the production stages, and that resources should only be acquired when needed for production. When discussing such inefficient procedural steps, the originators used the Japanese word *Muda,* translated as *futility*, *uselessness* or *wastefulness*, which furthermore highlights the importance of TPS in the refinement of this, at the time, new production methodology. Today, discussions and descriptions of *waste* still permeate the literature of Lean and Kanban.

Toyota faced other sources of waste than what the average software development team should expect to find, where the biggest source often was attributed to redundant stockpiling of inventory and resources. This is no longer the case. However, there are still semblances between physical production and software development, in that it:

1. Requires several steps to finish.
2. Relies on social entities and inter-organizational communication.
3. Entails a focus on reducing the total usage of resources to complete specific tasks.

The Lean mentality, perhaps even more in the general sphere of business management, retains an integral position of note. Throughout the field of software development, on the other hand, we often see that development teams and project managers tend to mix predefined procedures derived from both the Lean and Agile principles and frameworks (*see, for instance, Diebold et al., 2014 and section 2.1.4.2*).

Considering Agile development, we can see that it has its roots in more recent memory. Its inception is often accredited to a meeting held in Snowbird, Utah, early in the year of 2001. The participants of the meeting, thought leaders of the software engineering industry such as Kent Beck, Ward Cunningham and Ken Schwaber, had used this opportunity to discuss their concerns and frustrations towards the ineffective industry methodology-standard of the Waterfall model, and other traditional development techniques utilizing the concept of Big Design Up Front (Ambler, 2005). These frustrations and concerns, however, reached multiple decades back in time, and it was their shared sentiment that the industry needed a more "lightweight" methodology that could serve to reduce time-to-market for any software production company. They also argued that there was no such thing as a "finished product", and that enabling further development of any product line was becoming an issue of increasing concern. Spending two years of development prior to the software's initial release simply did not cut it in this brave new world.

As a summary of the meeting, the participants produced and signed off on the now infamous Agile Manifesto, promoting different values than what proponents of traditional methodologies had proposed. These included:

Individuals and interactions over processes and tools
Working software over comprehensible documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

Today, the Agile process framework *Scrum*, as discussed in section 2.1.4.1, has a leading methodological position (Diebold et al., 2014). The main resource about the framework is deemed the Scrum Guide (Schwaber et al., 2016). This resource consists of an in-depth description of each sub-part of the framework, complete with meeting schedules, team member roles, events, and so on.

Interestingly, there is only one single mention of *waste* in the Scrum Guide. There, it states, in relation to Sprints, that one should not be *"... allowing waste in the process"*. No further clarification of its meaning is produced, leaving the reader to infer their own. Instead, the resource promotes its users to *"[...] improve, within the Scrum process framework, its development processes and practices to make it more effective and enjoyable in the next Sprint ..."* As such, we see that their focus is not directly on improving the efficiency of a company's value stream, as is the case within Lean (to be discussed in section 2.2.3), but that becoming better adopted to the Scrum framework would imply the same increase in efficiency. *Unoptimized processes* could therefore be understood as a concurrent Agile definition of waste.

The concept of waste thus becomes somewhat blurred when investigating contemporary software development: Coping primarily with the Kanban method and the Scrum process framework, neither being well-founded nor, arguably, adequately defined, creating a highly complex set of diverging organizational constructs that we would try to probe with the same concept.

Waste as a *practical* concept therefore has been underdeveloped in the field of Software Engineering and urges a closer examination. I would argue that waste is a valuable and informative concept when discussing process-improvement within a single-case, organizational construct. This understanding of waste is also in line with the way the Toyota business managers used *Muda*; the useless, futile and wasteful behaviour we, the organization, needs to rid of, regardless of our structure.

2.2.2. Research

One paper that explores sources of waste in Kanban specifically is Ikonen et al. (2010), detailing a case study confirming the following sources:

1. *Partially done work*
    a. Delays in the critical path.
    b. Implementation of a given task required another, yet unimplemented task.
2. *Extra processes*
    a. Some of the Kanban routines were not necessary for small tasks.
    b. Inefficient Retrospective Meetings.

3. *Extra features*
    a. Members unaware of the key requirements.
4. *Task Switching*
    a. Two tasks rely on each other in the implementational phase.
    b. Disruptions: helping others or contributing to other tasks.
    c. Avoiding waiting: utilizing the time waiting by doing other tasks.
5. *Waiting*
    a. For customer representatives.
    b. For IT-support.
    c. For hardware shipments for the project.
    d. A person occupied when someone needs assistance.
6. *Motion*
    a. Lack of communication.
7. *Defects*
    a. Defects found at last minute.

These results are also overlapping with what is known as the *seven deadly wastes*, reported by Womack et al. (1996), to be discussed in section 2.2.3. The researchers studied an R&D – lab with 13 developers over a seven-week period. Here we see that the outlined waste sources are intended to be universal, meaning that these sources should be eliminated in any development team if found. Inefficient Retrospective Meetings were also mentioned as a repeating factor, indicating the need for an alternative or supplementing procedure.

Yet, identifying waste is still not a straightforward task. The problem often stems from the fact that we do not perceive such categories directly, but through the lens of the resulting output: We often do not understand that we are developing extra features, but we can, for instance, see that the team had been ignorant about the customer needs and intents following a Sprint Review. Being naive or ignorant about the customer's needs result in the production of the wrong features, or features being produced in the wrong order.

As we see, most of these points are multi-faceted to some extent, and contingent on the parameters of the organization. For instance, partially done work is perceived as a bigger problem in Lean methods than they are in Agile, as they create not only technical debt (Guo et al., 2016), but also so-called bottlenecks[4] in the task-flow. Which processes, tasks or events that are to be considered redundant is subjective at best, and one must consider an array of variables to make an informed judgement about them. Which features should be considered supplementary or redundant are first possible to identify when having a good grasp of the market you're developing for, as well as an understanding of the intention of the Product Owner whom ordered the software. Switching tasks might be essential for keeping the team moving forward in some cases, as to reduce waiting time for other members of the development team.

[4] If a team is unable to complete certain tasks using Kanban, they become unable to start the development of new tasks, because of Kanban's focus on limiting work-in-progress. Therefore, the "Doing" column of a Kanban-board might act as a "bottleneck", in that it limits the team's ability to effectively produce code. See Peterson (2009) for additional information.

Diverging organizations therefore perceive the waste categories put forward by Ikonen et al. (2010) in different manners, based on their business strategy, time constraints and organizational structure. The impact of waste on the development productivity is also intertwined. As such, identifying sources of waste amounts to identifying *possible* sources of waste, which again amounts to identifying *possible* areas of process improvement.

By looking at the literature on *critical success factors*, we can form such a picture, concerning the most crucial arenas for process improvement. Misra et al. (2009), for instance, showcases a framework that identified nine critical factors that have a *"[...] statistically significant relationship to "Success"*". These are:

1. *Customer satisfaction*
2. *Customer collaboration*
3. *Customer commitment*
4. *Decision time*
5. *Corporate Culture*
6. *Control*
7. *Personal Characteristics*
8. *Societal culture*
9. *Training and learning*

The study was conducted through a structured questionnaire answered by Agile practitioners. These respondents were thereafter included or discarded from the study based on eligibility criteria considering whether the respondents adhered sufficiently to the Agile principles, and whether they had previously been part of a traditional development team. 174 respondents were deemed eligible.

The framework represents the points of view that were addressed with a high frequency by Agile practitioners. But how does it relate to other sources on critical success factors? Here, we can see that the framework only partially agrees with the literature: Beneath, you will find a representation of fifteen relevant articles detailing *critical* or *common success factors,* set in light by the nine factors of Misra et al. (2010).

	Ahimbisibwe 2013	Bermejo 2014	Brown 2015	Darwish 2015	Hummel 2015	Kropp 2014	Matook 2014	Nord 2013	Pedersen 2013	Stankovic 2013	Tanner 2014	Tsirakidis 2009	Van Kelle 2015	Wan 2010	Wood 2013
Customer Satisfaction		■													
Customer Collaboration	■	■		■	■		■	■	■	■					■
Customer Commitment	■	■		■			■	■	■	■					■
Decision Time															
Corporate Culture	■	■	■	■	■			■		■	■		■	■	
Control	■			■	■		■	■		■		■			■
Personal Characteristics					■		■						■		■
Societal Culture		■			■		■						■		■
Training and Learning					■	■								■	

This set of articles was derived from a literature search for articles containing "Agile software development" and "success factors" that had been published after 2009, peer-reviewed, and which also included the latter search term in the title or the abstract. Some articles were removed by the researcher's leisure, as they were concerned with other fields of science, were too narrow in scope, or aimed at addressing only sub-parts of the success factors set (*such as, for instance, "social factors"*). It is not to be seen as a complete set, but should serve as a good indicator about which factors have been discussed with a relatively high frequency.

The matrix shows that there were few articles painting a broader picture of factors critical in ensuring success within an Agile development team. Hummel et al. (2015), Matook et al. (2014), Bermejo et al. (2014) and Wood et al. (2013) were the most overarching, and could possibly serve as introductory articles about probable areas of process optimization for searching project managers.

The fifteen articles also had a high citation-frequency for four other factors. These were:

- Planning *13 citations*
- Agile values *9 citations*
- Team structure *8 citations*
- Technical competency *6 citations*

By negating these factors, and removing factors that had less than three mentions, we can form a set of exhaustive waste categories that have been discussed with a high frequency within the literature. This set thus includes:

- Lack of planning
- Inadequate adherence to Agile values
    - Members lacking proper Agile traits
- Improper technical competency
- Lack of customer collaboration
- Lack of customer commitment
- Inadequate corporate culture
- Lack of control
- Unwillingness to train and learn
    - Inadequate promotion of training and learning

In addition to the waste categories of Ikonen et al. (2010), discussed previously in this section. This set of waste categories will serve as the basis upon which Research Question I will be evaluated against. When referring to case-specific instances of such categories (or potentially *new,* previously undefined categories), I will apply the term "*waste-objects*".

2.2.3. Definitions

There are many definitions of waste; some more specific in nature, while others painting a broader picture. One of the latter, coined by Womack et al. (1996), defines waste as primarily 1. "*... any human activity which absorbs resources but creates no value*". To clarify what such types of activities were, they produced a list now known as the *seven deadly wastes*, also including:

2. Mistakes which require rectification.
3. Production of items no one wants so that inventories and remaindered goods pile up.
4. Processing steps which aren't needed.
5. Movement of employees and transport of goods from one place to another without any purpose.
6. Groups of people in a downstream activity standing around waiting because an upstream activity has not delivered on time.
7. Goods and services which don't meet the needs of the customer.

In addition, Womack et al., (1996) differentiate between two distinct types of waste, termed *Type One* and *Type Two Muda*.

Type One Muda: *Steps, events or procedures that do not create value but are unavoidable.*

(*Usually*) developers don't write perfect code every time, and bugs may appear. Code revisions and bug fixing do not immediately add value, but cannot be neglected nevertheless.

Type Two Muda: *Steps, events or procedures that do not create value and are immediately avoidable.*

For instance, having geographically distributed development if it could easily be avoided.

To combat the amount of *Muda* in any organization adhering to the Lean principles, the most important task is to specify the value stream in relation to the customer ultimately buying, leasing or otherwise attaining the product that your organization is tasked or have chosen to produce. The value stream is considered as the "*specific actions required to bring a specific product* [...] *through the three critical management tasks of any business*." (Womack et al., 1996). These three tasks include the problem-solving task (finding out what the product is supposed to be), the information management task (receiving an order of the product, attaining and activating involved assets so that production can start) and the transformation task (producing, from raw materials, the finalized, shippable product).

As such, we can provide an alternative definition of waste, based upon its relationship with the value stream of a company: *Waste is all the human activities that hinder an effective and efficient value stream.* However, this definition, albeit more reliant on traditional Lean literature, is dependent on a value stream analysis, something that not all companies either perform or have any relationship to.

Instead, we would like to provide a definition that is more general than the latter, and still usable for both Lean and Agile development teams alike. With this foundation, we can define waste as the collection of:

1. *Unnecessary work*
2. *Incorrect work*
3. *Time spent on work that is not valuable*
4. *Time spent on work that is valuable, but at the wrong time*

What we can take with us from this definition is that processes and events can be ordered on a continuous scale from *valuable* to *not valuable*. It is not a binary relation; a process can be valuable, but not optimized. An event might be highly valuable, but inefficient. Taking this into account, I would argue that the *elimination of waste* from the Lean perspective only solves half the problem, as it rids of the work closer to *not valuable* (Type Two Muda), and keeps only that work deemed valuable: A remnant of its production line background.

Agile development solves the other part of the problem: It looks at the processes and events a team is enrolled in, and promotes continuous change and self-inspection as optimization techniques, for instance by usage of Retrospective Meetings. Some of these processes and events might be impossible to remove altogether, but might still be possible to further optimize. For instance, these might be inefficient meetings with a customer, or getting the customer to frequently update their requirements and prioritization. These are classic Type One waste-objects. As such, I would argue that expanding on our definition of waste would empower the software development community to better understand what is valuable for their organization, and what is not. Therefore, when I am discussing *waste* in this master's thesis, I will also include *processes and*

events that are not optimized as a defining characterization of waste. Doing so enables us to use the to-be-proposed technique on Scrum teams as well, as one of the focal differences between Agile and Lean development is exactly how process optimization is carried out.

As is pointed out in Diebold et al. (2014), nearly no team in the software development community are purists when organizing their working habits and procedures. Instead, we see that project managers, Scrum Masters, team leaders and so on tend to mix procedures derived from Scrum, eXtreme Programming, Kanban, DevOps and other Lean and Agile development methodologies together, shifting them about, and coming up with relatively unique patterns, working habits and procedures. By letting waste elimination also encompass process optimization, we are in a better position to discuss cross-methodological challenges that most developers and organizers can relate to. As such, this thesis will view waste as the collection of:

1. *Unnecessary work*
2. *Incorrect work*
3. *Time spent on work that is not valuable*
4. *Time spent on work that is valuable, but at the wrong time*
5. *Processes and events that are not optimized for the parameters of the organization or team in question.*

2.3. Identification Techniques

The primary driver of waste identification within Agile development is the concept of Retrospective Meetings. In addition, *Development Velocity* has become a measurement of project efficiency that many teams have chosen to utilize. In this section, both concepts will be discussed briefly, as they hold importance for later parts of the thesis.

2.3.1. Retrospective Meetings

Usually, waste identification is handled by Agile development teams by holding Retrospective Meetings. This event constitutes a cornerstone of Agile development, and serves as one of the principal facilitating elements that make an Agile development team tick, enabling them to improve their way of collaborative work. However, these meetings are only loosely defined and no framework encapsulating such meetings have been described and empirically tested to reduce waste, to the best of my knowledge.

The Retrospective Meetings, or *Sprint Retrospectives*, are of unique interest for this thesis, as this is the meeting where most discussions about possible process-optimization (and thereby waste identification) take place. The meeting is attended primarily by the development team, who discuss what went well and what did not in the preceding Sprint. It is this meeting that serves to foster continuous improvement within a team. Per the Scrum Guide, the motivation and purpose of the Sprint Retrospective is to:

1. *Inspect how the last Sprint went with regards to people, relationships, processes and tools;*
2. *Identify and order the major items that went well and potential improvements, and,*
3. *Create a plan for implementing improvements to the way the Scrum Team does its work.*

The goal of this meeting is to understand what is possible to improve upon, as well as planning for how to improve it. This plan is acted out in the subsequent Sprint, hopefully resulting in better structured procedures and processes, as well as a team that is able to provide more value to the customer, at less expense.

However, both researchers and practitioners alike point out that these meetings are often too long, insufficient and lack narrative (Almeida, 2014), and specifically so in distributed or large development teams (see, for instance, Hossain et al., 2009). Additionally, this meeting might constitute the only arena where team members have the possibility to vent their frustrations and concerns. As an outlet for frustration, it does not scale well, considering that less resources can be spent on each issue.

Poor management of group dynamics is another aspect that can damage the value of Retrospective Meetings. Ensuring that those who have something to say gets their voices heard is important, just as restraining those that like to say too much. This is a narrow path to traverse, and setting an agenda ahead of such meetings would arguably limit the impact of such dynamics. One way to do this, is to employ an in-advance data-gathering technique named Retrospective Wall Boards.

2.3.2. Retrospective Wall Boards

There has been little research on whether traditional Retrospective Meetings are effective at identifying the problematic areas of development procedures and processes. However, one technique that has been described and discussed by Agile practitioners is Retrospective Wall Boards (or, more simply put, Retrospective boards) (Almeida, 2014); giving the meeting some form of parameters.

The idea of a Retrospective board is to have a poster hanging in a public space, which is divided into three columns: Positive observations, negative observations, and actions that could possibly remedy them. The members of the development team are then asked to put post-it notes in each column once they experience something that they believe to be valuable for Retrospective discussions. If something is perceived as positive, the members put a post-it note in the positive column describing the observation. If something is perceived as negative, the members put a post-it note in the negative column describing the observation. In addition, the members can choose to describe an action that could help relieve the team of such negative observations in the future. They may also provide their name on the observations and actions, or remain anonymous if they so wish.

This board is actively filled out during the Sprint, and the observations and actions form the basis of what is discussed in the subsequent Retrospective Meeting. Here, the team collectively choose

which actions are to be taken, and uses the next Sprint to implement the changes that are agreed upon.

Retrospective boards are thought to make members of a development team more engaged in this important, albeit sometimes dull process. It also alleviates the members' memory, such that observations are stored immediately and does not have to be recalled some unspecified amount of days after the observation was first made.

This is interesting, as such a technique could lay the basis for the required data-gathering needed to perform a full-scale Process Tracing analysis (see section 2.4). Having such time-stamped observations would make it possible to trace what instances were reported when, and thereby understand the causal relationships between observations. This possibility will be further explored in section 5.1.3.2.

2.3.3. Development Velocity

Measuring how well a team is performing within Agile development has been an on-going subject since the formulation of the Agile Manifesto. One measure that has been broadly acquired by the field is called Development Velocity; intended to measure how much work has been done. Considering the estimated User Stories (see section 2.1.3.2), it is possible to view just how much could be completed within a Sprint. If the velocity (total amount of work conducted in a Sprint, based on "Story points" [5]) (VersionOne, 2017) increases, it signifies an increase in productivity (and possibly the elimination of waste), while a plunge in velocity signifies a decrease in productivity.

Measuring productivity is unfortunately a highly complex task, and the Development Velocity measurement is unsatisfactory in some ways. For instance, if the team members have little experience with estimating User Stories, they are prone to provide faulty estimations. One could argue that this issue would be reduced over time, but it does, however, persist whenever working with previously unexplored technologies, or when encountering other sources of uncertainty. At the current rate of which novel technologies are made available, it seems like this issue will persist for the foreseeable future.

This uncertainty creates fluctuations in the Development Velocity over time, resulting in a measurement containing little to no value for the team. The development might also be impacted by under-performance of third-parties, such as lack of involvement or response from the customer, or other service-providers not prioritizing the specific tasks being produced. These aspects lend evidence against Development Velocity as a measure of *actual* productivity, and it performs even poorer trying to measure how optimized the current development procedures are. Yet, it is still a measure that is utilized broadly in the industry, albeit requiring some expert knowledge about both estimations, as well as the development-techniques used.

[5] Relative points of measures intended to signify a "chunk of work", often established in terms of hours.

2.4. Process Tracing

Process Tracing is a set of qualitative research methods this is often used within single-case studies. Historically, it has its roots firmly planted in the Social Sciences, and tries to go one step further than simply establishing correlations between actions and outcomes. Its aim is to prove a stronger claim, by establishing and exposing *causal mechanisms*.

Is it so that democracy leads to peace? Or how did law X, posited by a Republican, pass congress when it was controlled by Democrats? These are the types of questions that Process Tracing could be a good fit for investigating. In the former, we see a possible Process Tracing study trying to establish the accuracy of a causal mechanism; namely, that democracy leads to peace. In the latter, we see a possible Process Tracing study aiming to unearth *which* causal mechanism(s) lead to a law getting passed. These are concocted examples, but should serve to highlight the types of studies where Process Tracing is and could be used.

As such, we can understand Process Tracing as a high-level analytical framework for analysing and investigating claims of causality, and the methods substantiate or discard these claims by referral to Causal Process Observations (CPO): Pieces of evidence substantiating or undermining a hypothesis. Collier (2010) states that "*Process Tracing examines diagnostic pieces of evidence - often understood as part of a temporal sequence of events or phenomena - with the goal of achieving and refining causal inference.*" This causal inference, or mechanism, is the driver of a Process Tracing study: Which X caused Y to happen?

The set of research methods that is Process Tracing, resembles traditional detective work, in that it often tries to understand why something happened after the result has been made clear. For instance, Beach et al. (2013) drew the analogy between Process Tracing and the classic Sherlock Holmes story "The Adventure of Silver Blaze", a short-story detailing the disappearance of a race horse and the death of its trainer. There, Mr. Holmes devised different tests to substantiate his hypotheses, and subsequently narrowed down the possible murderers to one single individual. These tests were contingent on the known facts (*i.e. the pieces of evidence substantiating or undermining his hypotheses*), creating a basis upon which hypotheses could be evaluated against each other.

All usages of Process Tracing share the same overarching principle: Looking at diagnostic pieces of evidence in a temporal setting, to infer all causal links between actions and outcomes. However, the goal of using Process Tracing might differ from study to study, as is highlighted in the two examples above, and some differentiation between its different sub-methods is therefore needed. Such a segregation has unfortunately been neglected historically.

2.4.1. Three sub-methods

Beach et al. (2013) state that *"The result of treating process-tracing as one method is a set of murky methodological guidelines, along with confused students and practitioners".* Instead they argue that we are better off differentiating PT into three distinct variants, based on their relative

purpose. All three proposed variants share some of the same theoretical foundations, yet deviate from each other on certain points. The authors present a list of such points, and take a set-theoretical approach to defining each sub-method listed in the forthcoming paragraphs. As such, the differentiating elements of the three variants are:

1. *Whether they are theory-centric or case-centric designs.*
2. *Whether they aim to test or build theorized causal mechanisms.*
3. *Their understanding of the generality of causal mechanisms (from systematic mechanisms expected to be present in a set of cases [population] to case-specific mechanisms).*
4. *The types of inferences being made, where theory-testing or -building variants makes inferences about the presence/absence of a mechanism, whereas explaining-outcome process-tracing enables inferences about the sufficiency of the explanation to be made.*
   (See Figure 2 for clarification).

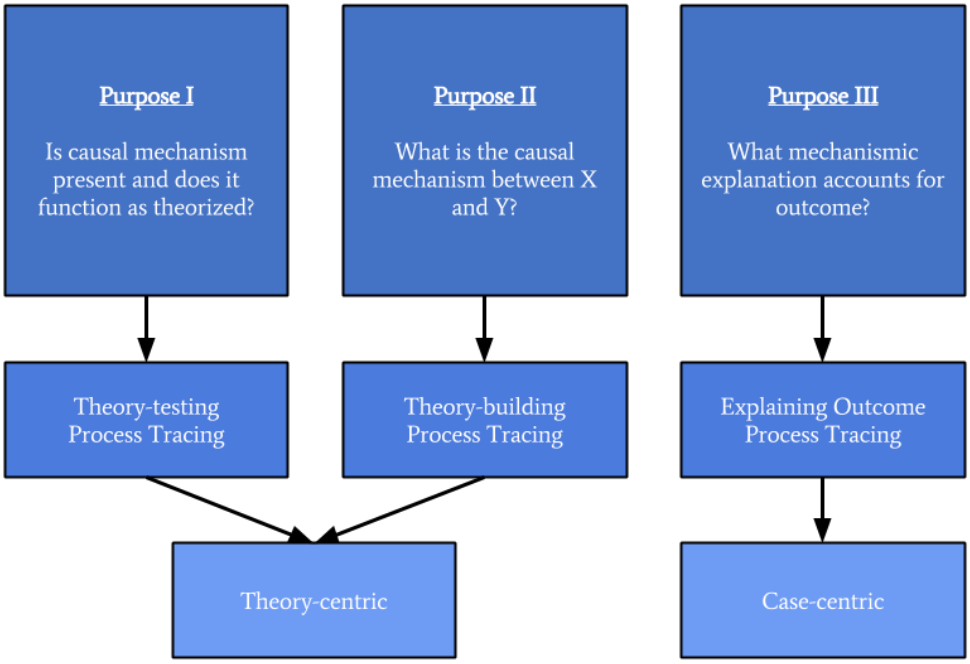

FIGURE 2: HIGH-LEVEL VISUALIZATION OF THE THREE SUB-METHODS.
ORIGINALLY PUBLISHED IN BEACH ET AL. (2013).

2.4.1.1. Theory-testing Process Tracing

Theory-testing Process Tracing (TTPT) is an analysis where the main goal is to investigate whether a previously hypothesized causal mechanism exists *in a specific case*. Here, the preliminary theory *(for instance, "democracy leads to peace")* is viewed as the causal mechanism, and the CPOs are the elements that lends evidence in favour of (or contradicting) the theory being tested. This method is often used when previous research has concluded with a strong correlation between X and Y, and a causal relationship seems probable.

The theories that are being tested could come from a variety of sources. For instance, if one was to conduct a Process Tracing analysis investigating why a software development project failed,

one might derive some possible hypotheses from the literature describing common factors in IT project-failures. One example of such a hypothesis might be that the project lacked proper management. One could also create a corpus of such common pitfalls, which might uncover additional hypotheses. Thereafter, the researcher might query the participants of the project about their own experiences and suspicions about why the project terminated as a failure, looking for clues supporting or undermining the hypothesis that one aims to test. The point is that theories could come from anywhere, and it is the researcher's goal to investigate whether the specific theory that one is testing is confirmed in relation to the CPO's that is recorded in the specific case, and that it functions as theorized. If it is not, some other theory should be tested.

As such, we can see that, with regards to the differentiating elements described above, this specific sub-method aim at testing theorized hypotheses, which, depending on where the theories are derived from, could be universal in their hypothesized scope, and can only be used to make inferences about the absence or presence of said hypotheses in a specific case.

However, since a specific Agile or Lean development team hardly have any *research* gathered about their own working habits, they also do not have any compelling evidence correlating their working habits and their results (*other than that of, possibly, Development Velocity)*, making the TTPT approach seemingly infeasible for our purpose.

From my understanding of this sub-method, the only conceivable way of using TTPT in our domain of inquiry would be to test for case-specific instances of waste heuristics, such as the list of waste categories provided in section 2.2.2. However, this would only constitute an understanding of the presence or absence of these categories, information that might not be sufficient in understanding the complete set of waste-objects in a development team without many iterations. Therefore, it seems less than optimal for our purpose.

2.4.1.2. Theory-building Process Tracing

The second sub-method is Theory-building PT (TBPT). Here we take an inductive stance, looking at outcomes as the first step to hypothesize about possible causal mechanisms which might have an impact on said outcomes. Beach et al. (2013) defines two possible situations where theory-building PT is a viable option:

1. *When we know that a correlation between X and Y exists, but we are in the dark regarding potential mechanisms linking the two (X-Y centric).*
2. *When we know an outcome, but where we are unsure what are the causes (Y-centric).*

At first glance, we might think that TBPT is a good fit for our purpose. For instance, a team might find themselves in a situation where they know the results of the previous development period, but not why it resulted in that specific manner (*Y-centric*). They might also have a good understanding of their organizational structure, but still not understand which mechanisms had implications for the result (*X-Y centric*). However, there is a discrepancy between the level of generality that TBPT entails, and the generality that Process Tracing as a Retrospective tool would have. The aim of a TBPT analysis is to produce a causal mechanism that is *universal in scope*, which is to say that the resulting causal mechanism of the analysis should be applicable for

all development teams. We have categories, heuristics and success factors informed by multiple case studies for this purpose, and by so, applying TBPT in this domain would also be redundant.

2.4.1.3. Explaining Outcome Process Tracing

Finally, the third sub-method is Explaining outcome PT (EOPT). Here, the main goal is to *"...craft a minimally sufficient explanation"* (Beach et al., 2013) of why an outcome occurred. This sub-method is often used to examine some historical outcome(s) from a single-case. Beach et al. (2013) also argue that although this method is highly case-specific, it can sometimes have implications reaching further than the specific single-case in question.

However, there is a distinction as to how one views the causal mechanism in this instance, which deviates from that of TTPT and TBPT. In those instances, the causal mechanism is deemed atomic, which is to say that there should only be a single causal mechanism that influences the outcome. This causal mechanism should either be the theory that one strives to test with TTPT, or the theory one is building with TBPT. On the other hand, in an EOPT analysis, the causal mechanism is inherently conglomerate, meaning that any causal mechanism that explains a specific outcome should be the conjunction of all variables influencing the outcome. Usually, a specific outcome cannot sufficiently be explained by one single variable (*i.e. inefficient process or procedure*), and thus needs to cope with the minimal conjunction of such variables sufficient in establishing causality.

Beach et al. (2013) notes that this is the most widely used instance of PT in the field. Concerning the prescribed scope of the analysis method, we also see that taking a single-case, non-generalizable approach would be much more in line with the Retrospective technique I propose. After an examination of all the three sub-methods described above, I deemed this variant of Process Tracing the most interesting for the identification of waste. Therefore, EOPT will serve as the theoretical foundation for the Retrospective Process Tracing technique.

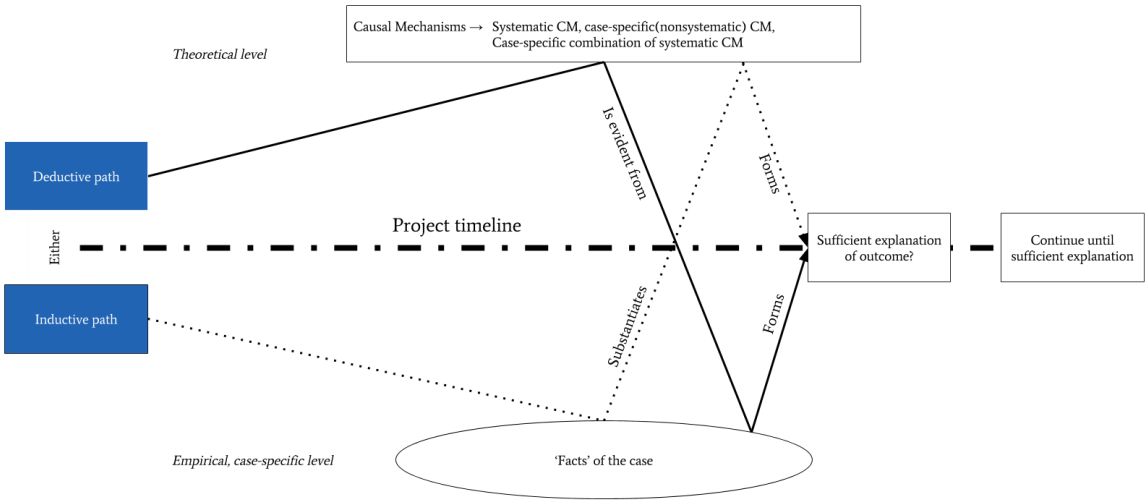

FIGURE 3: VISUALIZATION OF EXPLAINING OUTCOME PROCESS TRACING. ORIGINALLY PUBLISHED IN BEACH ET AL. (2013).

Beach et al. (2013) describe the workflow of an EOPT-analysis by way of Figure 3. As we can see, EOPT is discussed on two distinct levels: One that is evident, the empirical level, and one that is hidden, the theoretical level.

The empirical level represents the facts we know about our specific case: Facts that could be extracted from field observations, transcribed interviews, etc. The theoretical level, on the other hand, represents the "realm" where we can gather pre-theorized hypotheses: Literature, previous correlation studies, and so on. In our case, this could be the waste categories described in section 2.2.2.

We can work from both of these levels; either inductively or deductively. If we choose the inductive path, we work from the facts we know about the case to form a conglomerate description of which variables were important for the outcome. This is a method closely resembling traditional detective work. Here we know the outcome of a sequence of events, but do not have any direct information about which events influenced the outcome.

For instance, if we are to investigate a murder, we know the result of the causal chain (that a person has been murdered), but we are in the dark concerning how and by whom such a murder was carried out. However, there might be indicators (*blood on a sleeve, footprints matching that of a suspect*) lending evidence towards that it was, in fact, *Suspect I* who committed the crime. Having enough such incriminating evidence would enable the detective (*i.e. the researcher)* to form a minimally sufficient explanation about the question of guilt. This is a setting that development teams often find themselves in, where the murdered represents the poor result, and the waste-object represents the murderer; a defect, broken routine or suboptimal procedure that the team have great interest in remedying.

In either case, we are looking for a minimally sufficient explanation of the outcome, and thereby understanding the underlying, possibly conglomerate causal mechanism that lead to the outcome. Each of the hypotheses that we have theorized is then tested by utilizing the standardized testing-set discussed in section 2.4.3, to discard all hypotheses that fail those tests. When only one hypothesis remains, we have a plausible explanation of a causal mechanism.

If we choose the deductive path, we work from the basis of some theory. In this instance, the theory would be the hypothesized causal mechanism that resulted in some poor result. Here we look at how the team worked to substantiate our claims about a specific causal mechanism. To exemplify, this causal mechanism could be formulated as such:

$W(y)$ = *y is a result*
$P(x, y)$ = *x causes y to happen*

Axiom : $\forall y\,(W(y) \rightarrow \exists x P(x, y))$ = *For all Y, where Y is a result, there exist an X which caused Y to happen*

Possible values for x and y:
A = Insufficient planning
B = Schedule overruns

We take it that W(B) holds true. This means that there is some x where P(x, B). Is it then the case that P(A, B)? (*Meaning that insufficient planning caused schedule overruns to happen?*), and is P(A, B) a *minimally sufficient explanation* of the outcome? This is a contrived example, but should serve to exemplify the nature of a hypothesized causal mechanism. If enough evidence states *insufficient planning* as a repeating factor, we might, with varying degrees of certainty, form a minimally sufficient explanation: P(A, B); the project was indeed not planned well enough, causing schedule overruns.

2.4.2. An analytical framework

From the preceding, we can see that Process Tracing can be defined by its acute focus on uncovering, or theorizing, causal mechanisms. However, these mechanisms do not appear by themselves, but must be substantiated in some way. Therefore, Process Tracing is also referred to as an analytical framework, aimed at testing such hypotheses against each other. The CPO's (*e.g. evidence, observations, clues*) that is gathered throughout the study must be subjected to some form of analysis, to check whether they agree with the hypothesis one is investigating.

2.4.3. Four tests

There are multiple ways of going about this hypothesis testing, but some leading researchers on the matter (Bennet, 2011; Collier, 2010; Beach et al., 2013) all propose to use four tests; originally described by Van Evera (1997), as the framework of choice. Subjecting a hypothesis to these tests helps the researchers understand to which extent any certainty about the causal mechanism can be ensured, and enables them to discard hypotheses that have been shown to be inaccurate.

2.4.3.1. Straw in the Wind

The first *(and least powerful)* test is the *Straw in the Wind*. This is a type of test where the idea is to gain some insight rather than lending credence or direct evidence in favour of a specific hypothesis. An example of a Straw in the Wind test can be explained by the help of a bicycle theft: Was *Suspect I* in dire need of either quick cash or a bicycle? Passing the test suggests that our hypothesis of *Suspect I* being the culprit of the crime might be correct, but failing the test does not alleviate any doubt about the guilt of *Suspect I*.

2.4.3.2. Hoop

The *Hoop* test is a stronger test, where the researchers aim at either affirming the relevance of a hypothesis, or to be able to disregard the hypothesis completely. Let's take the same example of the bicycle theft: Was *Suspect I* in the same city as where the crime happened at the time of the incident? If that is the case, then *Suspect I* should still be further investigated. If *Suspect I* was not in the same city, then it is safe to discard the hypothesis about *Suspect I* being the culprit of the crime.

2.4.3.3. Smoking Gun

As the name implies, this is an even stronger test compared to the Hoop and the Straw in the Wind test. Here, we can be certain that the hypothesis is correct if the test is passed, but we cannot safely disregard it if the test happens to fail. Back to the bicycle-theft: If *Suspect I* was video-taped while cutting the bicycle's chain lock, then we can be certain that *Suspect I* stole the bike. However, learning that *Suspect I* was not videotaped stealing the bike, we could still not disregard him as a suspect.

2.4.3.4. Doubly Decisive

The final test, called *Doubly Decisive*, is the strongest, where passing the test implies that the hypothesis is right and all else are wrong. Failing the test implies that the hypothesis can be discarded. Unfortunately, the bicycle example comes short here. Creating such tests is hard in general, but the same effect might be achieved by combining a set of other tests, such as a Hoop and a Smoking Gun test.

2.4.4. Method of elimination

One way of selecting the perceived right hypothesis is by the method of elimination: If you, for instance, have three competing hypotheses, you do not necessarily need a Doubly Decisive or Smoking Gun test to select the right one. Instead, eliminating hypothesis I and II by aid of a Hoop test might constitute an adequate basis to make a judgement about the correctness of hypothesis III.

Having eliminated all other options, the last remaining hypothesis should be the correct one, however improbable. This is both a strength and a weakness of the framework. It is a strength that we can filter out the improbable hypotheses by usage of weaker tests, but one should be wary when dabbling with such techniques. As is always the case when working in an open world not directly bound by the laws of mathematics, we do not have complete knowledge about the world we operate in. There might be clues that we missed, hypotheses not uncovered, and so on.

2.4.5. Test boundaries

There are unfortunately no clear boundaries concerning what constitute a Straw in the Wind test and what constitute a Hoop test, what constitute a Smoking Gun test, and so on. Here, the researchers conducting the testing must base the tests and their perceived strength on some prior assumptions.

For instance, let's take the example of the bicycle theft again: If *Suspect I* was confined to a wheelchair at the time of the incident, it would be fair to argue that the test "is *Suspect I* physically capable of stealing a bicycle?" would constitute a Hoop test. Failing the test means that we could discard our hypothesis, namely that *Suspect I* was the culprit. However, if *Suspect I* was

a repeat offender and had extremely good upper body strength, was in dire need of money, and so on, we might not be able to assume *Suspect I's* innocence after all, making the test a Straw in the Wind test. Such cases arise more often than one would imagine, and especially so in cases with many CPO's.

2.4.6. Causal mechanisms

How causal mechanisms are viewed, as well as what they are, have been the subject of on-going academic debate in the social sciences (Hedström et al., 2010). Although I will not dedicate much effort in participating in this debate, there are a few points to mention that bears importance for the thesis.

One of these stems from Rohfling (2012), who argues that the current definition of Process Tracing comes short in a variety of scenarios. These scenarios are related to how we currently view causal mechanisms, and the hypotheses they constitute. Rohfling (2012) argues that the currently accepted research-wide definition of Process Tracing is too narrow, as it does not account for the differentiation between *realized* and *anticipated processes.*

2.4.6.1. Realized and anticipated processes

When researchers write about hypotheses in relation to Process Tracing, one could argue that they are really trying to describe the *why* of the resulting outcome. It is the job of the researcher to traverse the intermediate activities from the outcome of a situation until one uncovers the "triggering event" of the outcome. In relation to realized processes, this triggering event is the sole culprit, and is sufficient in establishing a minimal explanation of why the specific outcome occurred. In such cases, we are right in assuming a causal chain between the triggering event, the intermediate activities, and the resulting outcome, but this is not the case with anticipated processes.

There are certain scenarios when the triggering event really is a possible exogenous cause, i.e. an event that *might* happen outside the scope and control of a development team. The idea of anticipated processes and exogenous causes is that in such scenarios, we have a break in the causal chain between the event that might happen and the resulting output, since the triggering event is at the point of the intermediate activities still an eventuality. Therefore, we lose the assumption of causality and delve into the realm of possibilities. In such cases, we can only critique the argumentation of why the development team or its members chose to perform action X instead of Y, in relation to the anticipated process, and whether that action should be deemed wasteful behaviour.

This is a weakness of Process Tracing, and specifically so when applying it to the domain of software development. There are situations where strategic planning is contingent on anticipated processes: For instance, the prioritization of the Sprint backlog is dependent on external stakeholders, such as in cases where the development team are waiting for feedback or decisions made by the customer, the management, or other third-parties involved in the product being developed. There might be times where such exogenous causes impact the development pace

negatively, without the development team having any remedying power in the matter. Should these exogenous causes still to be perceived as waste?

I would argue that such cases should be exempt from being defined as waste, even though they contribute negatively to the Development Velocity. It is a fact of nature that we cannot accurately perceive the future in all situations, and that probabilities are at play in any strategic planning when not all facts are possible to access and assess. This should also serve as a critique against using Development Velocity as a standardised measure for assessing the progress in overall waste reduction within a project.

Unfortunately, there is no any clear-cut way of providing undisputable evidence that the actual development pace has increased (*i.e. more functionality produced at shorter time intervals*), as such anticipated processes interfere. I assume that this is one of the reasons why the measure of Development Velocity might appear erratic, if seen in the light of a small number of iterations. This at least holds true in teams with many stakeholders, or with team members without a long track-record in software development. Therefore, estimation techniques regarding Development Velocity seems inherently relentless and wasteful.

3. The Case

This section will serve to highlight the most vital information about the participating team that was involved in the study. This includes information about the product they were developing and its involved technologies, the company's situation and its organizational parameters, their development processes, as well as a brief profile of each of the participating members.

3.1. General

The study was conducted with a start-up company I was working with. The development team consisted of in total six individuals, five of whom were master students in Information Science, and one who had completed a bachelor's degree in Computer Engineering. In a usual week, each of the individuals spent on average 20 hours developing a travel expense reimbursement system. The team were co-located two full days each week, and otherwise worked from distributed locations.

During the study, a part of the system was in the process of being ported to iOS from Android, and the data gathering period coincided with the start of a new in-house project: The development of a web application intended for use by approvers of employee-incurred company expenses, and accountants responsible for the companies' financial systems.

This new application would enable companies to verify registered reimbursement claims, discard them, or add comments about certain receipts or expenses. The companies, that were intended customers, would also have the possibility to add external accountants situated in accounting firms, to handle the transfer of their reimbursement claims into their respective finance systems. The system described was relatively large in scope, and continued through and beyond the data gathering period.

3.2. Technologies

The system being developed contained several interfaces, where the actual reporting of expenses was to be made available through a mobile application developed for the Android OS platform. Multiple other technologies were used, such as Angular for the frontend development (which *transpiled TypeScript to traditional ES6 JavaScript*), a .NET-based C# project for the backend, a RESTful API also written in C#, as well as both Java and Objective-C for the Android and iOS applications, respectively. There was also a myriad of libraries used for web, mobile and backend development. The applications used SQLite for their long-term storage, while the server ran a MySQL database. All the backend code ran on a Windows server. In addition, the team had to use

different techniques for the structure and styling of the different layouts, where CSS3, HTML5 and XML formed the basis.

3.3. Situation

Prior to and during the data gathering period, the company were having multiple discussions and negotiations about integration-opportunities with other finance systems and service-providers. Such agreements would have provided the needed revenue stream for the company's long-term survival, as it directly influenced the number of users and companies that would acquire the product. This situation entailed a necessity of prioritizing such discussions and their prior planning, at times, over development. It also entailed that there were difficulties with prioritizing development tasks correctly, since there was a consensus that the development of functionality should be based on which agreements were struck, which was often hard to assess a priori.

Being a start-up company with limited funds, there was no any clear divide between the development team and the organization. In fact, the whole organization was, to varying degrees (see section 3.6), participating in development. Because of the company's situation, each of the developers also had additional responsibilities reaching further than that of coding and design. The company also had to consider legal matters, marketing, customer relations, contract negotiations, and so on, complicating the ability to effectively produce code.

3.4. Organization - Processes & events

The company used neither strict Scrum nor Kanban, but rather something which could be classified as Scrumban (see section 2.1.4.2), signifying the mixing of processes and procedures derived from both. At a higher level, one could view the company as a *Lean Start-up* [6], meaning that its focus was to develop a validated and researched *Minimally Viable Product* (MVP), or the minimal product that could credibly establish a revenue stream for the company. To understand what this MVP was, it became important for the company to also hold regular meetings with potential stakeholders in the product, which furthermore took time away from development.

The team structured their processes iteratively, but their development continuously. This means that the processes, such as Planning and Retrospective Meetings, were held on a two-week basis. These two weeks constituted a Sprint. However, there was no defined Product Owner, so the company saw no need to estimate user stories, or provide estimations for cost of development. This enabled the company to develop the system continuously.

[6] See Blank (2013) for clarification on *Lean Start-Up* and *Minimally Viable Product*.

During the Sprint, the team held the following collective activities:

Week 1:
Tuesday:
- *Scrum Planning Meeting*
- *Co-located development*

Thursday:
- *Daily Stand-Up Meeting*
- *Co-located development*

Week II:
Tuesday:
- *Daily Stand-Up Meeting*
- *Co-located development*

Thursday:
- *Co-located development*
- *Retrospective Meeting*

3.4.1. Daily Stand-Up Meeting

During the Daily Stand-Up Meeting, the team discussed briefly what each developer had done since the last sprint, what they planned to do, and whether they had any impediments that would require assistance from other developers. This event was time-boxed for fifteen minutes.

3.4.2. Scrum Planning Meeting

During the Scrum Planning Meetings, the team laid out a high-level plan for which part of the development should be focused on during the next Sprint. No definite *goals* we set in terms of new functionality, User Stories or engineering tasks, but the meeting focused on creating a shared vision about the *prioritization* of work for the next iteration. This manifested itself in the Sprint Backlog which highlighted a quick-and-dirty prioritized queue of functionality, tasks or features which should take precedence during the next two weeks. The event was time-boxed for 1,5 hours, but often required less time.

3.4.3. Retrospective Meeting

In the Retrospective Meetings, the team tried to understand what went well in the previous two weeks, what did not, and how one could improve upon these things. However, it had historically been used for discussions relating to strategy planning, or different opinions considering

technology choices, etc. This event was time-boxed for two hours, but frequently exceeded this limit, due to the nature of the discussions that were brought up.

3.5. Development Processes

The development process was, as stated, continuous, following the pull-principle of Kanban (see section 2.1.4.2). This meant that the team members chose which tasks to develop themselves, but oftentimes this became self-explanatory because of the amount of involved technologies (see section 3.2). There was, unfortunately, quite a divide between the knowledge-bases of the different developers, and this had become more apparent lately, when some aspects of the system were completed, while others were still very much in the making. On the other hand, the team members did choose which engineering tasks to be developed when, as long as it was in alignment with the shared prioritization that had been established in the Sprint planning meetings.

To ensure the completion of tasks, the team limited the amount of work-in-progress, and visualized and tracked the workflow by utilizing a tool named Trello [7]. This tool enabled us to perceive which phase an engineering task was in, by looking at the "Trello-board" (e.g. a digital Kanban Board, see section 2.1.4.2) made available for the team. This board was segmented into the following columns:

- Product Backlog
- Development
- Testing
- Done

If a task was chosen by a developer, the developer moved that item to the Development-column. When done developing, the item would be moved to the Testing-column. Here, the relevant involved developers performed functional testing to check if the item was ready for production. If the item was deemed acceptable, the item would be moved to the Done-column and employed. Such items were often pair-programmed[8], and at least so when it required communication between different technologies or parts of the system.

The team had chosen to implement a simplification of both the "cross-functionality" and "collective code-ownership"-principles of eXtreme Programming, where the team had designated a lead developer for each technology, and a secondary developer that had different tasks, but should be technically capable of aiding on difficult, or large items. See section 3.6 below for clarification.

[7] A digital tool for organizing tasks. See www.trello.com for more information.
[8] More information about XP and its principles can be found in Wells (1999).

3.6. Team Member Profiles

Five developers shared a common education, in that they all had completed a bachelor's degree in Cognitive Science and were enrolled in a master programme in Information Science. They also applied for much of the same courses during their studies, such as Artificial Intelligence and Software Engineering, but had diverging interests in software development otherwise. These are highlighted below.

Lead Frontend: Designated lead developer frontend, as well as product and interaction designer. Lead Frontend also had a keen interest in project management. Because of this, he was designated as the Scrum Master of the project.

Lead Backend: Designated lead developer backend. Lead Backend also had an additional responsibility as lead developer for our mobile applications, and had an interest in Artificial Intelligence. He had also been programming since his youth, and had expert knowledge about most of the technologies that the organization chose to utilize in this project.

Secondary Frontend: Secondary developer frontend, as well as responsible for frontend-backend communication. He was also involved with sales, customer communication and external meeting planning.

Secondary Backend: Secondary developer backend, having a main responsibility in creating the necessary API-functionality. He too had additional responsibilities as our in-house accountant and contract negotiator.

Lead Marketing: Contributed to interaction design, but were mainly involved with creating marketing material, in addition to handling social media profiles and customer relations.

Lead Security: Lead Security had completed a bachelor's degree in Information Technology, and was the only developer that had not shared an education with the rest of the participants. He had previously worked with Lead Frontend, and had wide knowledge about web security, object-oriented programming and the HTTPS-protocol. His main responsibility was to embed security measurements, handling confidential customer information correctly, and setting up the deployment-routine and server-based software.

This team was deemed a good fit for the study, as they aimed to be able to make a living off the income the company generated when they finished their Master's theses. Identifying waste was therefore seen as a direct return on investment, ensuring that the participants were both motivated and interested in helping to refine the to-be-proposed framework.

4. Methodology

Researching a novel application of a research method presents a set of problems that needs to be addressed, some of which are usually not seen in other studies of Information Science. First and foremost, it is important to note that Process Tracing is *not* the research method used in this study; rather, there is a new layer of methodology that is used to study the participating members' usage of Process Tracing.

To adapt Process Tracing to a waste identification technique, we need a method that allows for flexibility. Experiences and knowledge gained throughout the study holds unique value for the research, and can help substantiate or discard any thoughts that had incurred prior to the data-gathering.

Where the research proposed should be scrutinised, the application of Process Tracing must be pragmatic. By this, I mean that we are mainly interested in manipulating or altering the Process Tracing method into a usable piece of organizational "theory", that can be applied by practitioners. We are not interested in preserving all aspects of Process Tracing, in any way. If the research finds that Process Tracing is lacking in some respects, regarding its waste identification capacity, we are better off altering it, making it better, rather than following all guidelines described by the method's proponents.

This chapter will describe which research method was used in studying Process Tracing as a Retrospective tool, as well as its possible alternatives and their shortcomings. In addition, I will also discuss relevant data gathering techniques for the chosen method, as well as any ethical considerations worth addressing.

4.1. Choosing Method

There are a few methods that are inherently well-suited for studying contemporary, real-life situations. The most applied qualitative research method used in the field of Information Science and Software Engineering, as of 2008, was case studies (Dingsøyr et al., 2008), singular or multiple. What such studies excel in, is studying already defined concepts. For instance, some research has been conducted using case studies in trying to understand the differences in implementation of Agile development teams (Diebold, 2014), and the phase that incur when an organization transitions from a traditional to an Agile organizational structure (Nikitina et al., 2012). Others have studied the relationship between development experience and the correctness of planning-poker estimations (Haugen, 2006). Speaking in broader terms, case studies are well suited for trying to understand the *how's* and *why*'s of some predefined concept accounting for all situational variables, over a given amount of time.

Initially, I decided to perform a case study with an external development team to assess how well suited Process Tracing was as a Retrospective tool. The need for a case study was obvious, since studying a development team was the only way to produce sensible data. However, engaging in a case study also implied that one should not interfere with the participating team in their work, but merely observe and deduce knowledge from the observed. This approach seemed too rigid, as one would have to have a complete theory of how to use Process Tracing as a waste identification technique in Retrospective Meetings, and educate the team on its usage, prior to the study, based on loose assumptions.

As such, it seemed as if using case studies was not a preferable approach when performing this type of exploratory research. We could not assign a high probability to Process Tracing working instantaneously as a waste identification technique, but rather asserted that some level of alteration was needed to optimize PT for this specific task. Therefore, I decided that performing a case study with an external team, without having the sufficient level of influence needed, would result in a poor research design, which could have rendered the research questions unanswerable. This type of method would be better suited to assess Process Tracing's generalizability as a Retrospective substitute or supplement, *after* it having become adapted by the industry. As with software development, the requirements of this tool were subject to change when new knowledge was gained through experience with its usage. A more flexible approach was needed, where the researcher could actively influence the participating team, as well as getting feedback from the team, in how they used Process Tracing as a Retrospective tool. Action Research seemed to be the only approach that allowed for this level of flexibility.

4.2. Action Research

Action Research is often historically linked with Kurt Lewin, who is believed to have coined the term (Mills, 2000). As is stated in Oates (2006), '*action research has been used particularly by professionals who want to investigate and improve their own working practices*'. This seemed to align perfectly well with the vision of this master's thesis, the goal of Process Tracing as a Retrospective tool, and the focus on process optimization in Lean and Agile development. As a research approach, it can be defined as "*...a participatory, democratic process concerned with developing practical knowing in the pursuit of worthwhile human purposes, grounded in a participatory worldview which we believe is emerging at this historical moment. It seeks to bring together action and reflection, theory and practice, in participation with others, in the pursuit of practical solutions to issues of pressing concern to people*" (Reason et al., 2001).

The approach is characterized by:

1. *A concentration on practical issues*
    a. Instead of dealing with abstract hypotheses, the emphasis is on real-world, practical issues that people face in their ordinary working lives. Waste, for instance, could be one such issue.
2. *An iterative cycle of plan-act-reflect*
    a. The approach is understood as a framework for planning changes, acting them out and reflecting upon the impact the changes made.

3. *An emphasis on change*
    a. In action research, the focal point is to actively engage in the issue being studied to provoke and encourage changes, and to continuously improve the concept, routine or process that is the subject of study.
4. *Collaboration with practitioners*
    a. The participants of the study are actively engaged and contribute to the research directly.
5. *Multiple data generation methods*
    a. Usually, more than one data gathering technique is implemented in Action Research studies. Triangulation is deemed important to uphold the judgement criteria of interpretive and qualitative research, such as *confirmability* and *dependability*.
6. *Action outcome plus research outcomes*
    a. Both practical and academic outcomes can be achieved by using Action Research. For instance, a team might deem the Action Research project a success, as it solved a real-life practical problem, but it might be of little academic value. Vice versa, a team could deem the project a failure, but the knowledge gained might still be academically valuable. Fortunately, in our case, action and research outcomes are one and the same, as we are performing an *academic evaluation* of Process Tracing as a Retrospective tool, that has *practical importance* for the participants.

The Action Research method is designed as a three-step, repeating cycle, where the goal of the research technique is to iteratively improve, or better understand, some concept, question or routine.

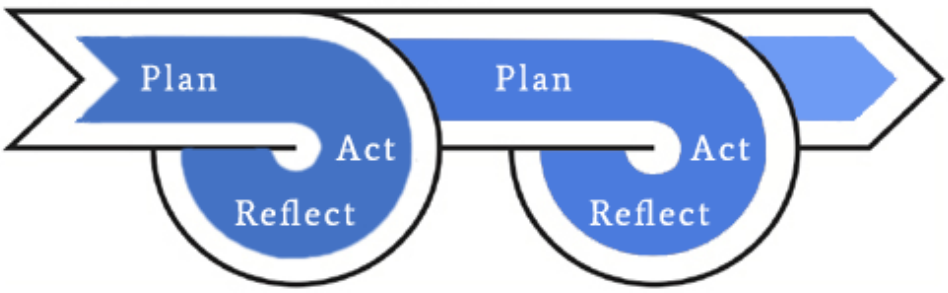

FIGURE 4: VISUALISATION OF THE ITERATIVE ACTION RESEARCH CYCLE.

There are some commonalities between Action Research studies that are symptomatic for the method. In greater detail, we can say that an Action Research study usually involves the following steps:

1. **Planning**:
    1.1. Identification of a problem.
    1.2. Developing a plan for solving the problem.
2. **Acting**:

2.1. Implementing a change.
2.2. Collecting data concerning the parameters of the new implementation.

3. **Reflecting**:
   3.1. Analysing the data and forming a conclusion about the success of the implementation.
   3.2. Modifying the theory.

- ✓ *Repeating step one through three until satisfied with the routine.*
- ✓ *Reporting the results.*

Action Research revolves around problem-solving. At the heart of each Action Research study, there is a set of specific problems, often of practical significance, that the researcher is interested in rectifying. As a research method, it distances itself from other research methods in that it does not explicitly enable generalizability of the findings, other than providing a solution to the problem in the very specific and limited domain the research project was carried out in.

This procedure can be exemplified by imagining a teacher wanting to increase her students' writing proficiency. Following the previous steps:

Identification: The students underperform in writing skills relative to their peers.
Plan development: Maybe the students' writing skills are contingent on the feedback that is provided after each test or homework assignment?
Data collection: Gathering relevant data about other teachers' students' grades, her own students' grades, prior to and after the iteration.
Analysis: Did the grades of the student become better after increasing the level of feedback for each test or homework assignment?
Theory modification: If no, choose a different hypothesis and perform a new iteration. If yes, then report the findings.

By using this framework, professionals and practitioners can improve their own routines and practices, contingent on evidence-based reasoning. Yet, it is not a strict framework, and does have its weaknesses as a full-blown research method, if evaluated as a positivistic research project.

The fact that very few methodological guidelines are provided might be one of the reasons why this method has not been used widely in any field other than that of pedagogy research. In addition, ensuring the generalizability of results of an Action Research study is hard, an obvious set-back for researchers considering the technique.

4.2.1. Critique against Action Research

Historically, it has been put forward different critiques against Action Research as a valid research discipline. Most have treated Action Research as a research method aimed mainly at teachers and professors, and therefore not all the critiques are relevant in this setting. However, some still need to be addressed. According to Hodgkinson (1957), these include:

1. *Action Research is only problem solving, and hence academically inadequate.*
2. *Action Research is statistically inadequate.*
3. *Action Research does not lead to defensible generalisations.*

Even though this critique is old, it still bears significance in the academic debate. The above points are admissible in our case, since:

a) We are not assuming Process Tracing as a Retrospective tool to be implemented successfully in all cases, but only in cases that are equal to the participating development team in parameters and scope. Given the work setting and the organizational parameters of the participating team, the findings would therefore still be of importance to a broad audience. Given the scope of a master's thesis, it is not viable to *both* provide a proof-of-concept of Retrospective Process Tracing, *and* test this technique within multiple cases with diverging organizational structures, team sizes and stakeholder relations. Therefore, we can assume the admissibility of critique 1 and 3.

b) Waste-objects are not statistically quantifiable, as there are far too many obfuscating factors at play. We cannot statistically measure waste, with the state of the research into waste at the time of writing, and neither is that the goal of Process Tracing as a Retrospective tool. See, for instance, the discussion of Development Velocity (section 2.3.3) and its shortcomings. Hence, we can assume the admissibility of critique 2.

One criterion, usually held in high esteem in studies of a positivist nature, is that the relationship between actions and consequences can be anchored in existing theory. Concerning Action Research, this means that the actions that are taken, based upon the evidence and data that is gathered, should be grounded in existing literature that is more general in nature than the specifics of the study at hand. For instance, if one action is perceived as a good approach to solve the Action Research problem, then the study will need to argue for that action by relating the solution to existing literature.

This case is also admissible in our situation, as finding proof in the literature is exactly what we aim *not* to do. If such proof is found for all uncovered waste-objects, we can provide a definite "no" as the answer to Research Question I.

By studying this problem, I also aim at solving a problem mainly faced by practitioners. This implies that using Process Tracing as a Retrospective tool *is* mainly a pragmatic, problem-solving exercise, setting this task apart from other studies aiming to understand and describe some concept, theory or practice in general terms.

4.3. Participatory Research

A point in favour of Action Research is that the participants of the study are all 'insiders'. This comes at the expense of objectivity, but also grants the study some unique traits that can only be found in participatory research. Paraphrasing Kemmis et al. (2014), who argue that:

1. Only participatory research can create the necessary conditions for the practitioners to understand and develop their processes 'from within', i.e. with the innate knowledge of what is being studied, and the understanding of the context in which the given problem is studied.

2. In participatory research, the participants can talk about the given problem being studied in a shared language, i.e. with a shared vocabulary about the problem domain.

3. Only participatory research enables the participants to directly participate in the development of new routines and processes, as a response to the problem, process or routine being studied.

4. Only participatory research enables the participants to participate in developing the communities of practice that is being studied, both between different stakeholders of a specific practice *and* between people collectively responsible for a specific practice.

5. Only participatory research can enable the participants to actively transform the conduct and consequences of a practice to meet the changing requirements of external pressure; specifically, when the consequence of the practice is:

    a. *irrational* (due to a lack of understanding of the practice),
    b. *unsustainable* (due to the practice being ineffective, unproductive or otherwise lacking the necessary optimization),
    c. or *unjust* (that the practice serves the interest of some, at the expense of others, or causes unnecessary conflict which could be avoided).

Action Research is a good fit for our purpose, as it is flexible enough to allow for inter-study modifications, and it also aligns perfectly with the iterative nature of Agile development methodologies. As such, the researcher can be actively engaged with a team of developers over a period, and it fits the setting that developers who are interested in using this tool would usually find themselves in.

The arguments of research design flexibility, the iterative alignment of the team's Sprint cycles and the Action Research cycles, the answers to the research method's critiques, and the capacity of participatory research should serve as the justification of why Action Research is a valid research approach in the given setting.

4.4. Qualitative Data Gathering

To ensure an appropriate level of credibility when performing an Action Research study, it is important to present the readers with enough information so that they can perform an 'audit trail' (Oates, 2006). This means that the iterative adaptation of Process Tracing as a Retrospective tool needs to be well documented, and there are several techniques we can use to gather the needed documentation. These include document analysis of meeting logs, video and audio logs of

Retrospective Meetings, interviews, as well as direct and indirect observations. In Action Research, the most relevant data gathering techniques include document analysis of meeting logs, direct observations, as well as *(structured and unstructured*) interviews with the participants.

4.5. Ethical Considerations

The purpose of ethics in research is to ensure that the research conducted is performed in a manner that does not negatively impact the participant's rights, self-image or person in general. In an Action Research study, the principles of confidentiality, anonymity, and informed consent are of importance. None of the gathered information in this study should infringe on the participants right to confidentiality or anonymity, and each of the participants were informed about how all the gathered information were to be handled and presented in this master's thesis.

4.5.1. Confidentiality & anonymity

Confidentiality refers to the principle that the generated information in an experiment or study should only be made available to those granted access (Fossheim, 2009). Anonymity refers to the principle that the participants of the study should not be uniquely identifiable. These points are essential in cases where the gathered data might have a negative impact on the participants if they get in the wrong hands, which could be the case in medical or psychological research. The gathered data in this specific study should not be of such a character, but the participants were made aware of what they were participating in. This was handled by providing the participants with a document stating that they were giving their informed consent to proceed with the research. Additionally, the Norwegian Centre for Research Data reviewed the proposed handling of data, prior to the study, and granted an approval to proceed.

4.5.2. Informed consent

Giving an informed consent means that the participants of a study are made aware of what information is stored in which manner, as well as how the research is conducted, and gives an informed approval to proceed with the research. In this study, it means that both the company and the participants are presented with the reason for the study, and are given the option to be treated anonymously. The former was approved through a board meeting, and the latter through individual notice.

5. Research Design

Having outlined both Action Research and the participating team, I will in this chapter specify how the method is to be used and implemented in this study. In addition, I will propose how to answer the previously defined research questions with this implementation, and describe how to gather the data on which to perform the analyses.

5.1. Overview

The study will be segmented into three phases. These include an initial *educational phase*, a *problem identification phase*, and a subsequent *Action Research phase*.

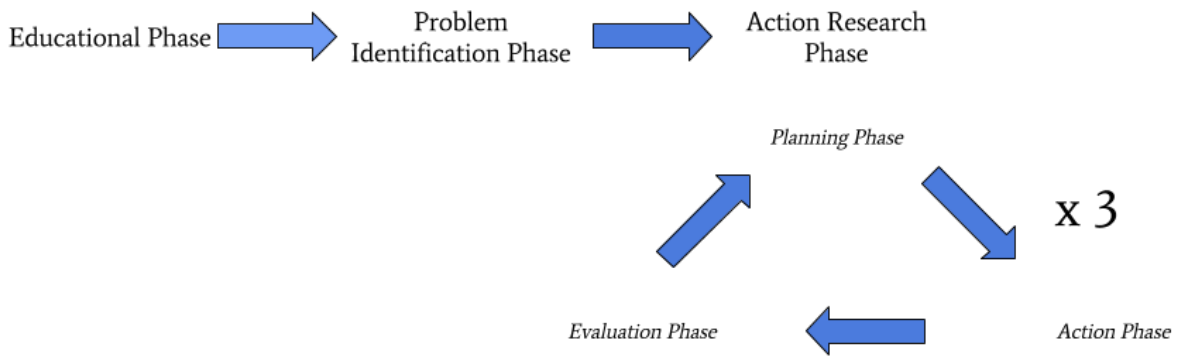

FIGURE 5: OVERVIEW OF THE STUDY PROGRESSION.

5.1.1. Initial educational phase

It was assumed that the development team would need some education about Process Tracing, Action Research, and other involved concepts that was of importance to this study, prior to the Action Research phase. This was handled ahead of the study, enabling a shared vocabulary between the participants in all subsequent discussions.

5.1.2. Problem identification phase

After the initial educational phase, the team members met again to discuss their problems and concerns towards their waste identification capacity as it were prior to the study. In this phase, the topic of discussion was divided in three overarching questions. These were:

Question 1: *Which perspectives of our current waste identification routine are perceived as positive aspects?*

Any positive aspects of the Retrospective Meetings that were mentioned in this discussion should be addressed and preserved by Retrospective Process Tracing.

Question 2: *What are your concerns about the Retrospective Meetings as of right now?*

Any obvious problems that were brought up by the team members should be addressed by Retrospective Process Tracing. If these are handled, we can at the very least ensure that the team have increased the value of their own Retrospective Meeting, regardless of the rest of the results, indicating a practical outcome of the study.

Question 3: *How can we improve our waste identification capacity without infringing on other tasks, such as development?*

Some of the team members may have had perspectives on how to improve or streamline the waste identification capacity of the team. If so, such perspectives would be valuable pieces of information that might provide guidelines for how to implement or improve Retrospective Process Tracing.

5.1.3. Action Research phase

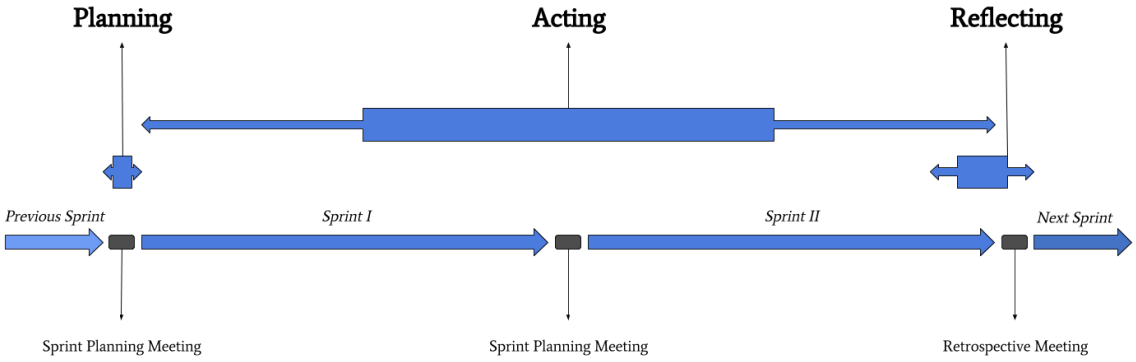

FIGURE 6: RELATIONSHIP BETWEEN ACTION RESEARCH CYCLES AND SCRUM SPRINTS.

Above we can see a high-level representation of the iterative alignment of the Action Research cycle and the Sprints. The cycle started with planning for changes, followed by acting them out, and lastly reflecting on whether the changes improved the practice at hand. As is evident, the Action Research cycle and the Sprints coincided well, enabling the researcher to improve the Retrospective Process Tracing technique from iteration to iteration.

The goal of this Action Research study was guided by both a practical and an academic purpose, grounded in the problem identification phase. From the participants' point of view, we were interested in maximizing the waste identification capacity of the team by ensuring that our new technique included the perspectives found in the problem identification phase.

From a researcher's perspective, I was interested in seeing whether Process Tracing was conceivably the framework that could help in solving these issues. I was also interested in investigating whether the framework could find waste-objects that were not directly possible to explain by looking at the literature on waste. These formed the basic, overarching questions for the Action Research study, which were iteratively performed over three four-week, two-Sprint, Action Research cycles.

5.1.3.1. Planning phase

Ahead of every 2nd Sprint, the team enrolled in a planning situation, where changes to the waste identification routine, processes and behaviour were discussed. The practical goal of these sessions was to analyse the data gathered from previous iterations, and produce a specific plan for new improvements to reduce waste. The academic goal was to use this data to improve upon Retrospective Process Tracing to better understand the waste that resided within the project.

5.1.3.2. Action phase

In the action phase, the goal was to implement the changes agreed upon in the planning phase. From a practical point of view, this meant that the participants of the study changed their processes and behaviour according to the improvement plan. From an academic point of view, the goal was to implement the changes proposed to augment Retrospective Process Tracing. The latter was up to the researcher to perform, while the former was understood as a collective task.

The action phase would also serve as the data gathering phase for the subsequent reflection phase. Here, I decided on a set of initial data gathering techniques that was to be used. These included:

1. Observations made and communicated by the participants

The participants of the study were asked to store observations that they thought could contribute negatively to the development pace in a spreadsheet. For each observation, the members stored:

- a description of their observation,
- the date at which they experienced it,
- their participant number,
- the situation in which they experienced the observation *(optional)*,
- and one or more hypotheses about what caused the problem *(optional)*.

This data-gathering technique resembled a Retrospective Wall Board, but went one step further by having the participants also registering contextual and sequential clues.

2. Direct observations of the Retrospective Meetings

The observations that were reported in the previous development period were to be discussed in the following Retrospective Meeting at the termination of every 2nd Sprint. There, the participants got the opportunity to discuss their observations and hypotheses, which might highlight possible new clues that could be used for further identification of additional waste-objects.

3. Discussions held between team members in open communication channels

Discussions held in open channels were also deemed a possible source of information about waste-objects. However, most of the specific development issues that the participants faced seemed, from a historical point of view, to be conveyed verbally from the person facing the issue to the lead-developer of that piece of code. Information that was not directly registered during the collective coding sessions were communicated through the communication tool Slack[9], which the team used to transmit all questions, problems and information about the project. These discussions were deemed a possible source of information. In addition, using such a tool was also seen as an enabler for finding a possible *lack* of communication. It was believed that, in some cases, this could also serve as an identifier of wasteful behaviour.

4. Research notes about causal process observations made by the researcher

A problem of Action Research is that the method relies heavily on the internal motivation of the participants. This was, unfortunately, not in the hands of the researcher to control. As such, the researcher received an extra burden in observing and storing observations, at least until the participants started producing observations on their own, as well as showing a willingness to discuss them in an open forum with the rest of the development team.

5.1.3.3. Reflection phase

In the reflection phase, the researcher would analyse the collected data with usage of a modified EOPT method (see section 2.4.1.3. and Figure 7). The practical goal of this phase was to investigate whether our proposed changes led to fewer observations, or at least fewer observations of the same nature as those that had previously been reported. The academic goal was to analyse whether the changes to Retrospective Process Tracing made it better at describing waste-objects in a manner that led to a higher actionable potential, meaning that it became better at describing waste-objects that either had a *new* categorical character or could be classified as a waste category previously addressed (such as those described in section 2.2.2).

The descriptive focus of explaining waste, rather than the traditional focus on causality between actions and outcomes, led to a modification of the EOPT-method. The initial version can be viewed in Figure 7.

[9] An online chatting-solution for teams. See www.slack.com for more information.

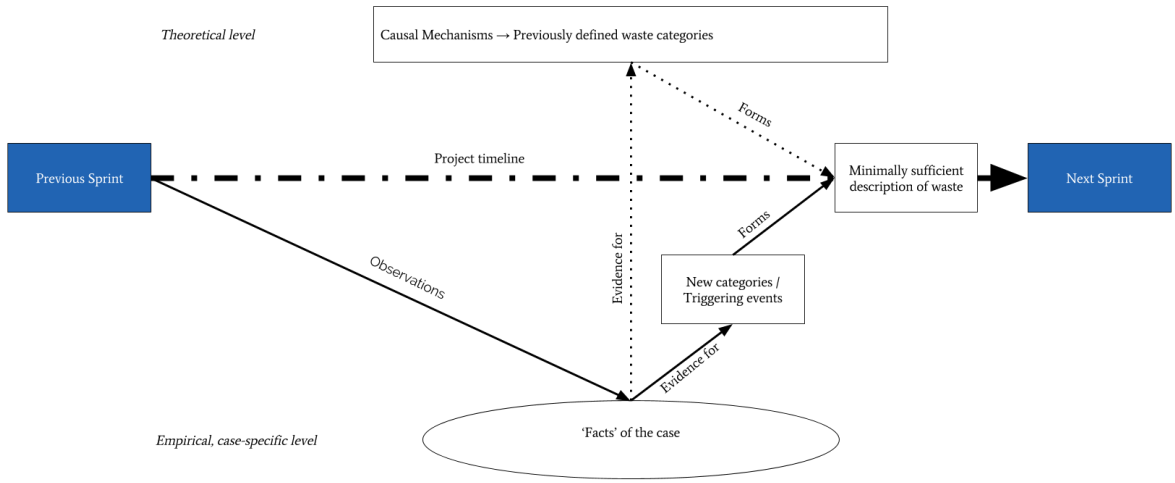

FIGURE 7: GRAPHIC VISUALIZATION OF THE MANIPULATED EOPT CATEGORIZATION TECHNIQUE.

Here we see that once new participant observations were registered, there were two options: Either the observations could be explained by the previously defined waste categories, or they would constitute evidence for new, case-specific waste-objects, or specific triggering events (see section 2.3.3), that were a direct result of organizational structure.

These categorical answers or previously theorized causal mechanisms, which were found evidence for, would conjointly serve as the *minimally sufficient explanation* (MSE, see section 2.4.1.3), which in this setting would be a complete, yet abbreviated description of the identified waste-objects that were found in the preceding Sprint.

The Action Research phase of the study was conducted for a period of three months, spanning three Action Research cycles and six Scrum process cycles (see sections 3.4 and 3.5). This would give the study a proper chance at producing a stable set of criteria for how to use Retrospective Process Tracing to identify waste within a team, as well as identifying unique waste-objects that were the direct result of the team's organizational parameters.

5.2. Answering the Research Questions

Here, it is again important to state that the research questions are to be answered in a non-generalizable way. Considering the research design and the exploratory basis of the thesis, we find ourselves in a position where we can make informed judgements about the research questions, but only to the extent of this specific team, and possibly those similar in parameters. By following this research design, I should be able to answer the research questions, considering the perceived scope, as follows:

Research Question I: *Is it possible to identify waste-objects not covered in the existing waste literature by using Retrospective Process Tracing?*

This question is answerable by looking at the *minimally sufficient explanations* that is the result of an analysis following the completion of every 2nd Sprint. If these minimally sufficient explanations describe some waste-object, be it an event, process or procedure, not directly visible from the waste categories defined in section 2.2.2, we can conclude that Retrospective Process Tracing does serve a unique purpose, in that it enables the identification of waste-objects that arise because of the uniqueness of the development team, and its situation. If not, we can report that Retrospective Process Tracing could only identify waste-objects that other tools and techniques (i.e. literature surveys) could unearth; making the technique redundant, albeit possibly more cost-efficient than a literature review.

Research Question II: *Can Retrospective Process Tracing be implemented with a relatively low initial cost?*

In the development team, the Retrospective Meetings were originally time-boxed to two hours. Having six individuals attend the meeting implies that the total cost of such meetings would amount to twelve work-hours not assigned to development (disregarding individual salaries). If we are, by usage of Retrospective Process Tracing, able to shorten this meeting by one hour, and not spend more than six hours on the analysis prior to the meeting, we have implemented a new technique that, in effect, saves financial resources; both in terms of boosting development time and, hopefully, increasing our waste identification capacity.

In any way, answering this research question amounts to performing a break-even analysis, considering the prior cost of Retrospective Meetings, and the cost of Retrospective Process Tracing. An analysis of time spent on both the Retrospective Meetings prior to and after implementing Retrospective Process Tracing, in addition to measuring time spent on the analysis and the registration of observations, should enable us to judge whether the technique saves resources, costs the same, or adds to the total cost.

Research Question III: *Is Process Tracing enough to serve as a substitute to traditional Retrospective Meetings, or should it be used as a supplement?*

The Scrum Guide promotes three goals for the Retrospective Meeting (see section 2.3.1). If Retrospective Process Tracing addresses these three goals, as well as any concerns uncovered in the problem identification phase, we should be able to classify Retrospective Process Tracing as a full-scale substitute to traditional Retrospective Meetings, and not just a supplement. If the three goals stated in section 2.3.1 are not met, we can conclude that Retrospective Process Tracing can only serve as a supplement to traditional Retrospective Meetings.

6. Data Collection

This chapter will seek to highlight the most valuable information and conclusions that could be drawn from each of the research phases, described in chapter 5.

6.1. Initial Educational Phase

Before the start of the Action Research study, the participants did require some education on the merits of both Process Tracing and Action Research.

This part of the study was conducted as two seminars, spanning over two days, each lasting approximately four hours. Here, the group of participants were co-located at the premises of the Department of Information Science and Media Studies at the University of Bergen. The development team members were largely unfamiliar with the methods of both Process Tracing and Action Research, so a considerable amount of time had to be spent initially to get them up to grips with the involved concepts. Having worked with Agile development techniques for close to two years prior to the study, the participants were well-versed in the concept of waste, and encountered no problems understanding the new definition that were laid to ground, defined in section 2.2.3.

Concerning Process Tracing, the sessions focused on the most critical concepts that were thought would require participation from their part, such as the four tests described in section 2.4.3, the idea of a minimally sufficient explanation described in section 2.4.1.3, as well as the agreed-upon data gathering technique in form of the digital Retrospective Wall Board (sections 2.3.2 and 5.1.3.2) used as the primary data gathering technique. In addition, a more thorough introduction to the concept of waste, hereunder Type One and Two Muda, were held to form a collective understanding of the concept, enabling a shared vocabulary on the matter.

6.2. Problem Identification Phase

After the initial educational phase, the team met again to discuss the problems of the current Retrospective Meetings, and how its waste identification capacity could be improved. This discussion was based on the three questions presented in section 5.1.2 and below. This session took in total one hour to complete.

Question 1: *Which perspectives of our current waste identification routine are perceived as positive aspects?*

There seemed to be a consensus on what function that Retrospective Meetings had within the team. One participant summed it up quite clearly when stating that "*Retrospective Meetings [...]*

serve the same purpose as therapy sessions" (Secondary Backend). Discussing previous development cycles, a point was made that the meeting had historically been used to settle arguments unrelated to waste identification (see section 3.4.3). This way of using the Retrospective Meetings is in a clear contrast to the Scrum Guide (see section 2.3.1), and might have been a result of the team's indifference towards the project members' roles. Most of the participating team members were not solely confined to solving tasks directly entailing production value, but also had responsibilities reaching further (see section 3.6). This made the Retrospective Meetings an arena for discussions that, per the Scrum Guide, should not be addressed in the Retrospective Meetings.

It was concluded that it was important to retain the openness for discussing peoples' frustrations and differing opinions in the new technique. One could argue that such differing opinions are sources of waste in themselves, as they might act as inhibitors for productive work, and should therefore be resolved.

Question 2: *What are your concerns about the Retrospective Meetings as of right now?*

Lead Backend stated that "*frequent changes in topics lead the discussion we are having astray".* This seemed like a legitimate claim, as the discussions often deviated into issues that were different from those initially brought up. Not being able to conclude a discussion often led the team to spend more time than strictly necessary on specific issues. This implies that the meetings became long, tedious and resulted in few actionable proposals for improvements.

As a response to the previous statement, Lead Marketing argued that our Retrospective Meetings lacked focus. The participants used the meeting not to the effect described in section 2.3.1, but for other discussions that were not fitted to be had in the rest of the meetings we were enrolled in.

The consensus reached implied that it would be practical to instigate some constraints to the meeting, such as distributing an agenda prior to the meeting. Such an agenda would also help those not directly involved in the issue at hand in understanding and contributing constructively to the discussion.

Question 3: *How can we improve our waste identification capacity without infringing on other tasks, such as development?*

As the concluding remarks, the team agreed that we had the potential to cut the allocated time of the meeting by becoming better structured in such Retrospective discussions. Therefore, we agreed that an agenda should be produced, highlighting the desired focus areas that should be discussed, and that we should retain the open discussions until we had covered these. In addition, a new strategy meeting was included in the iterative activities to limit the amount of unrelated discussions in the Retrospective Meetings. We also agreed that our Retrospective Meetings *had* to result in an actionable list of improvements.

These areas of improvements would serve to increase our waste identification capabilities, leading to more time allocated to development and fewer unrelated discussions. Therefore, the team agreed that a new Retrospective routine should be able to:

- *Provide an agenda setting the focus of the meeting, prior to the meeting.*
- *Reduce the total working hours spent on Retrospective Meetings.*
- *Retain some form of open discussion.*

Before the data-gathering period, I had originally planned to perform six Action Research cycles, each spanning one two-week Sprint, but lack of data prevented any meaningful discussion to be drawn from the data material following the first Sprint. This is unfortunately a weakness of Retrospective Process Tracing, in that it requires an active participation from the team members to be a useful technique.

This meant that we decided to only perform the Retrospective Meetings every other Sprint, as we had established a new meeting aimed at covering strategic business-decisions and their related discussions. Since we perceived the frequent and underperforming Retrospective Meetings to have little value, we decided that it would be better, for our case, to increase the time between these meetings. This would serve to reduce waste and increase the total hours of development that we had available. This also implied an elongation of the Action Research cycles, implying that they now would span two Sprints. Therefore, Figure 6 showcases the correct alignment of the Sprints and the Action Research cycles, while section 3.4 showcases our meeting structure as it were *prior* to this reorganization.

6.3. Action Research Phase

In this section I will showcase the data collected from three Action Research periods, their respective observations, and how I went about performing the analysis on this data. The section will be segmented into three parts, one for each iteration. Here, I will write about the patterns that were discovered in the data, the minimally sufficient explanation of the iterations' data set, their subsequent Retrospective Meetings, and any general insight that was gained about this specific usage of Process Tracing.

6.3.1. Iteration I

The first analysis was performed following a four-week, two-Sprint period. The analysis was conducted based on 16 observations, evenly reported among the development team members. All members provided at least one observation in this iteration, and no member reported more than four. The group was asked to report observations of events, procedures and situations that they believed contributed *negatively* to the development pace, but was not limited otherwise. The team were also not pressed in any way to produce observations. It was vital for the study's integrity that the development team members cited actual observations on their own terms, and not concocted observations to simply serve the research.

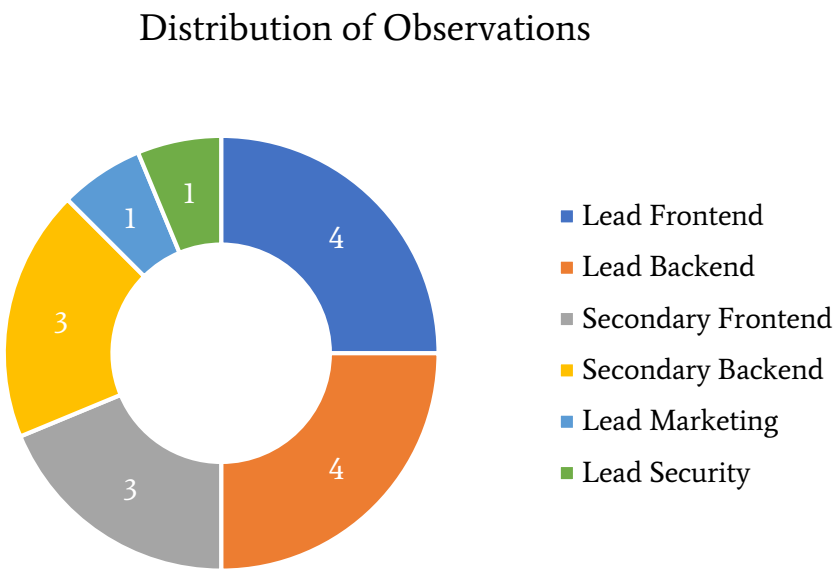

FIGURE 8: SPREAD OF OBSERVATIONS IN THE 1ST ITERATION.

Each observation was reported on the predefined format specified in 5.1.3.2. These observations were stored online, and the reporting scheme was always accessible by the development team members. The reporting scheme was also deemed, initially, sufficiently detailed to perform a Process Tracing analysis, as it coped with both the categorical (situation-stamped) and sequential (time-stamped) perspectives, that are of interest to this study. Beneath is the presentation of the total data set that formed the basis of the analysis.

[???] signifies fields not filled out by the participants, while XX/X/Y/Z denotes anonymized third-parties. All the observations were reported in Norwegian, and were therefore translated, to the best of my ability.

Observation Number	Date	Participant ID	Description of Observation	Situation (optional)	Potential actions (optional)
1	10.01	Lead Frontend	*Cannot continue development of web-app without test data*	Code development	*1. Increase priority of development tasks we know should be done before other peoples' tasks* *2. Create good test data before the start of development*
2	10.01	Secondary Frontend	*Meeting with XXX is needed before planning of web application development*	Code development	*Meet with XXX (Should've been prioritized earlier)*
3	10.01	Secondary Frontend	*Early focus on sales of a product that is not ready for the corporate market.*	Prioritization	*More frequent contact with potential customers would probably resolve the issue.*
4	12.01	Secondary Backend	*Cannot test new API-functions*	Code development	*Lead Security needs to be more available. We're lacking in HTTPS*

					competence
5	17.01	Secondary Backend	*Hard to develop functionality for people that isn't here*	Code development	*Have more time for being co-located*
6	19.01	Lead Marketing	*Problematic to get time for being co-located with all the diverging schedule and activities that people attend*	Planning	*[???]*
7	31.01	Lead Backend	*Lack of communication leads to half the people attending some sessions half the day*	Communication	*Clarify the time of meeting for collective activities can be performed way ahead of the meeting*
8	02.02	Lead Backend	*Saying that you're 15 minute late, for some, becomes 30+ minutes...*	Time	*Delay the «morning meeting»*
9	2.02	Lead Frontend	*Nothing of value is produced before all the team members are present, since the meeting is the first point on the daily agenda*	Daily Stand-Up Meeting	*Move the meeting to the end of the day, or become better to meet at the specified time*
10	2.02	Secondary Backend	*Full stop in development of integration with XXY, we lack documentation from their part*	Code development	*[???]*
11	2.02	Lead Frontend	*Lack of hardware, specifically Mac's, leads to only one developer being able to develop the iOS-application*	hardware	*Maybe think about getting a loan*
12	2.02	Lead Security	*I have no idea what this backend-code is, and no one is available when I have time*	Code development	*Having more people available*
13	2.02	Secondary Frontend	*External stuff (ref. XXZ) leads to poor prioritization of engineering tasks*	Cooperation with third parties.	*[???]*
14	9.02	Lead Backend	*Continuous delays make the meetings start later than planned for*	Meeting	*[???]*
15	9.02	Lead Backend	*Awaiting further development considering our collaboration with XXZ*	3rd party	*Wait*
16		Lead Frontend	*We started a new development project RIGHT at the start of a holiday for one of the essential team members*	Web development	*Thoroughly test the API before we start to add complexity with front-end logic*

By performing a categorization, I found relationships between most of the observations. These included:

1. **Lack of planning:** Seven observations (*1, 2 7, 8, 9, 12 and 16*) cited some form of poor (or absence of) planning.

2. **Lack of cross-functionality:** Three observations (*4, 5 and 16*) cited some form of *cross-functional development* (*multiple types of technologies required intercommunication between different developers, sometimes leading to waiting*).

3. **Lack of clear prioritization:** Four observations (*2, 3, 5 and 15*) cited *lack of clear prioritization*, leaving some team members without knowing what to best spend their time on.

4. **Lack of follow-up of approved proposals for improvements:** Four observations (*7, 8, 9 and 14*) cited that the team had failed in implementing specific proposals for improvement previously discussed.

There was not a clear one-to-one mapping between all the observations and their assigned categories, (considering observations 5, 7, 8, 9, and 16), and some were not covered by the categorization at all (observations 6, 10, 11 and 13). Two of these, 10 and 13, were related to work on an external product integration, where we lacked documentation concerning their API, and were unable to acquire it promptly. Observations 6 and 11 were unrelated to the rest of the data set.

The list of observations displays a clear relationship between *lack of planning* and *lack of follow-up*. Observations 7, 8 and 9 denotes similar observations, but were reported by two individuals. These observations simply denoted an inability to follow up on actions that had been decided on prior to the data gathering period, relating to our agreed-upon proposals for improvement to our planning routine.

6.3.2. Iteration II

The second analysis was performed after a new four-week development period, where the development team wanted to see a decrease of observations similar in parameters to those categorized in the previous iteration.

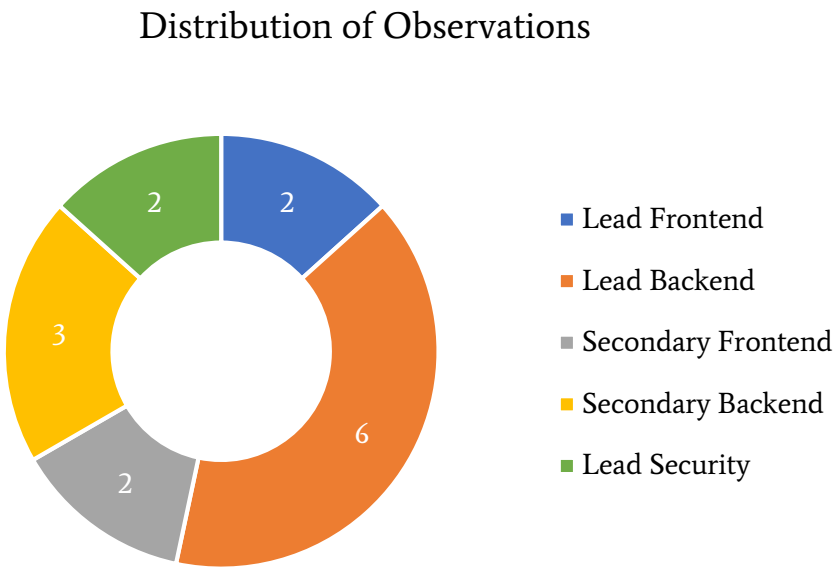

FIGURE 9: SPREAD OF OBSERVATIONS IN THE 2ND ITERATION.

Here we can see a less distributed set of observations, containing in total 15 reports. In this iteration, one participant did not provide any observations, while another provided six. This was unexpected, but serves to highlight some interesting points worthy of discussion. Firstly, the one participant who provided no observations (Lead Marketing) responded that he had not encountered any problems that caused a decline in his development pace. Lastly, Lead Backend, who provided 6 observations, had registered three distinct observations (CPOs 22, 23 and 30) citing the same problem (*but at separate times*), namely that it was a distinct lack of participation from some members he needed to work together with.

Observation Number	Date	Time	Participant ID	Description of Observation	Situation	Potential actions
17	16.02	03:15	Lead Backend	*Inadequate quality of and lacking documentation on existing code*	Code development	*Consider refactoring*
18	21.02	12:40	Lead Backend	*Lack of progress with XXX*	Meeting activities	*Follow up the communication more thoroughly*
19	21.02	14:30	Secondary Backend	*Reduction in work days leads to inefficiency because of many "loose threads" from day to day – unable to complete anything*	General work settings	*Consider working more effectively every other week, with more than two workdays those weeks.*
20	21.02	15:46	Secondary Backend	*XXX never returned with an answer for our request.*	Establishing meetings	*Be more proactive with evaluating our need for contact with third parties, as well as being better at following up our requests*

21	21.02	22:00	Lead Security	*Still no answer from XXX*	Don't know what to code	*[???]*
22	21.02	10:06	Lead Backend	*People absent on short notice.*	Communication	*[???]*
23	23.02	10:13	Lead Backend	*People absent on short notice again.*	Communication .... Or what?	*[???]*
24	27.02	21:00	Lead Backend	*Poor code creates a mismatch between the Android application and the API*	Code development	*Consider refactoring*
25	28.03	10:00	Secondary Backend	*People don't show up*	Communication	*Some people need to realize that we are waiting on them if they're not at the meeting.*
26	02.03	16.14	Secondary Frontend	*Work as seldom as we do makes getting back to grips with the code tedious*	Prioritization	*[???]*
27	02.03	22:00	Lead Frontend	*Scrum seems to be a problematic framework for our situation*	Specifying methodology	*Change method to Kanban or DevOps...*
28	07.03	10:00	Secondary Frontend	*2 persons with fixed tasks are usually not present, leading to lack of progress in their respective areas*	[???]	*[???]*
29	07.03	04:00	Lead Frontend	*Lack of response from XYZ lead to a lot of work writing certain documents at the last minute*	[???]	*[???]*
30	09.03	10:00	Lead Backend	*Still lack of participation*	Planning meeting	*Arrange the meetings ad hoc. Those who depend on each other should coordinate between themselves.* *Are people lacking in motivation, or are there other things that are playing in?*
31	14.03	21.25	Lead Security	*Deliverance of needed resources took 1,5 months instead of the assumed two weeks*	Awaiting hardware shipments	*[???]*

By performing a similar categorization to that of the 1st iteration, we can describe some patterns in the gathered data:

1. **Substantial time-constraints:** Three observations (19, 26 and 27) expressed problems with the project's scope and time-constraints.

2. **Third-party collaboration:** Four observations (18, 20, 21 and 29) expressed negativity towards the slow pace of discussions with third-party service providers.

3. **Poor coding standards:** Two observations (17 and 24) cited that poor coding standards, or differences in standards between developers, lead to a lot of time being spent understanding code that had not been written by the participant reporting the observation.

4. **Lack of motivation:** Five observations (22, 23, 25, 28 and 30) were indicating that some participants had become uninterested in the project. This could also be categorized as *lack of planning*.

From this iteration, we can see a clear one-to-one mapping between the reported observations and their assigned categories. These categories are also covering the full data set. Observations from two categories had previously been reported: *Lack of motivation* (under the name of *lack of planning)* and *third-party collaboration*. However, *substantial time-constraints* and *poor coding standards* were new observations that had not previously been reported by the participants.

6.3.3. Iteration III

The last iteration was also performed after a four-week period, yielding 11 registered observations.

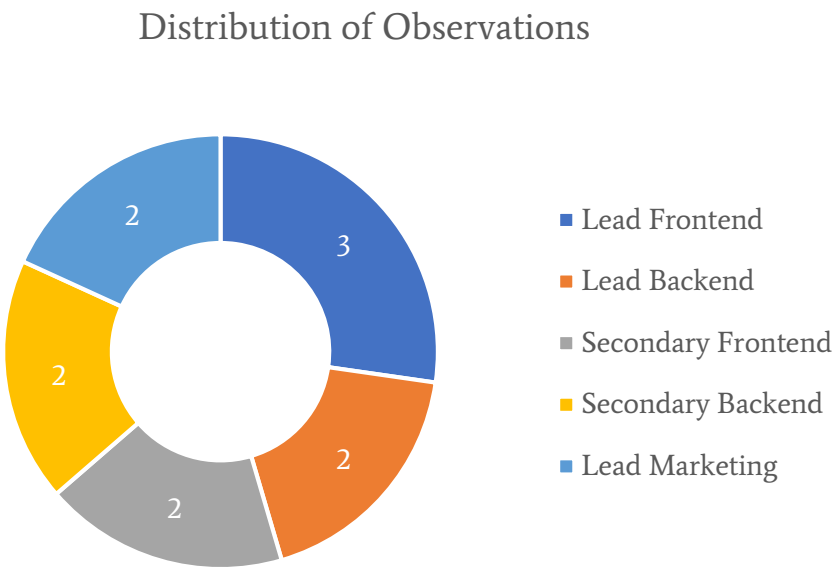

FIGURE 10: SPREAD OF OBSERVATIONS IN THE 3RD ITERATION.

Here, we found a more evenly distributed data-set, indicating that no one experienced any significant or frequently recurring sources of waste.

Observation Number	Date	Time	Participant ID	Description of Observation	Situation (optional)	Potential actions (optional)
32	18.03	23:00	Lead Backend	*Writing master's thesis takes away time from development*	Any	*[???]*
33	18.03	01:20	Secondary Frontend	*Other tasks (such as writing the application or communication with XXX) took focus*	Any	*[???]*
34	25.03	10:34	Lead Backend	*Still some technical debt. This will take some time to fix*	Code development	*Refactor all code to desired standard*
35	27.03	14:15	Secondary Backend	*People have generally less time because of their theses*	Any	*Focus on completing theses, enabling more time for development*
36	29.03	11:48	Lead Frontend	*Web-app development is not prioritized because of the focus on integrations*	Coding sessions	*Use time for front-end developers to enhance their backend-capabilities?*
37	02.04	21:00	Lead Frontend	*Versioning of the Angular project is under pair. Different developers have different installed versions of node. Frontend-project doesn't compile*	Committing to the Git-repository	*Create a plan for implementing new versions to the frontend-project*
38	04.04	10:04	Secondary Frontend	*Updating Angular leads to other dependent libraries breaking*	Pulling from the Git-repository	*[???]*
39	04.04	13:12	Lead Frontend	*Lacking API-functions*	Coding Sessions	*Either assign me to other tasks or prioritize the frontend/backend-communication higher*
40	06.04	15:30	Lead Marketing	*Have to work with different assignments at the same time*	Any	*[???]*
41	10.04	21:10	Lead Marketing	*Implemented certain function on the landing page wrong. Might lead to fewer page views and poorer Google-rank*	Evaluating Google Page-rank (Marketing)	*Defect found after launch. Should've tested these things in some way...*
42	11.04	11:45	Secondary Backend	*must evaluate other contractual offers because of questions about the projects short-term viability*	Long term planning	*Ensure funding so we know that this project is worth participating on.*

The final categorization uncovered three distinct categories:

1. **Substantial time-constraints:** Four observations (32, 33, 35 and 40) cited some form of lacking the necessary time to produce code efficiently.

2. **Poor technical solution:** Two observations (37 and 38) cited a problem relating to frequent updates to the frontend framework Angular and its dependencies, which we used for parts of the system.
3. **Poor distribution of resources:** Two observations (36 and 39), also relating to the frontend development, cited inefficient coding sessions as the backend-crew were preoccupied and could not produce the requested new functionality.

This mapping encompassed eight of in total eleven observations, where the remaining three observations did not seem to be either connected with each other or the remaining data set. These highlighted the following issues:

1. The company website had a faulty function which needed to be sorted out.
2. Uncertainty about the short-term viability of the project.
3. There was still some legacy code that needed to be revisited sooner, rather than later.

Nr.1 could be exempt from the definition of waste, but served to highlight some lack of testing prior to the launch of new marketing materials. This might have negatively impacted the *Page-ranking* [10] of the webpage, and at the very least showcased an instance of Ikonen et al. (2010)'s category *defects*. However, it was a simple error to sort out, and Lead Marketing fixed the issue within ten minutes of debugging.

Nr. 2 highlighted that there was one member who brought into question the short-term viability of the project, as the developers were about to finish their master's theses and needed pay. Measures had been taken, but this might have not been communicated clearly enough to the whole team. Therefore, this was deemed a non-issue, and could not be classified as waste.

Nr. 3 seemed like an indication that the whole code-base had not yet adopted the prescribed design patterns and code standards (this is detailed in the next chapter. Specifically, see sections 7.2.4 and 7.2.6), but measures had also been taken to alleviate this concern. In the previous Retrospective Meeting, we had decided that the code should be revisited and reworked to a specified set of coding standards and design patterns, but this would take a while to complete (see section 7.2.6).

[10] A measure of how «important» a website is; counting the number and quality of links from external sources. Adams (2016) provides an in-depth discussion on the measure.

7. Analysis

This chapter will be segmented in three sections, coinciding with the iterations that were described in sections 6.3.1 through 6.3.3. In each of these sections, I will highlight the important conclusions that could be drawn from the gathered data material, both in terms of the practical waste-objects that were found, and the academic merits of Retrospective Process Tracing and how it was further improved.

7.1. Iteration I

This section will discuss the four categories that were found in the data-collection:

1. Lack of planning
2. Lack of cross-functionality
3. Lack of clear prioritization
4. Lack of follow-up on approved proposals for improvements

7.1.1. Lack of planning

From the offset, we could see that the four uncovered categories of the first iteration were only loosely overlapping with the findings of Ikonen et al. (2010). Although lack of planning was not specifically mentioned in this source, there are numerous others that indicate that overall project planning is key in enabling rapid, Agile development. From our set of waste categories specified in section 2.2.2, thirteen out of fifteen sources mentioned planning as a critical success factor. For instance, Brown (2015) pointed at the initial *project definition process* as an important first step in creating a unified understanding of the problem domain, both *between* the customer and the development team, but also *within* the team. Kropp et al. (2014) found that *clear project requirements* was another important success factor.

What both these sources take for granted is that there is an external customer, someone who is paying for, and has certain expectations of, the system being developed. Unfortunately, in the case of Lean Start-Ups (see section 3.4), there isn't always any specified customer that ultimately is paying for the development, which means that the initial requirements are dependent on a limited (and continuously expanding) understanding of the project scope. This understanding matures as new pieces of information are discovered, and new, previously unknown groups of stakeholders emerge. These types of projects entail that the planning needs to have a certain nimbleness for rapid change, and that long-term planning is unfortunately a commodity seldom available.

Lack of planning is the category I would deem the most interesting in the first iteration. The lack of planning was not directly visible, or something that any single developer observed, but the

category became apparent when considering the whole of the data set. For instance, one member of the development team observed that there was a need for more information about a new module that was going to be built. To substantiate or discard our hypotheses about the customers' needs (observation 2), an interview with an external accounting firm should have been conducted. Another team member observed that development of that same module could not commence before more test-data had been provided, which were at the point hard to generate; because of key developers being ill or having to prioritize other parts of the development (observation 1). This case was possible to both comprehend and plan for ahead of time. Unfortunately, the team did not, resulting in an inferior performance. The development of systems dependent on either test data, or the actual database, which at the time were not fully implemented, therefore suffered.

In total, this shows that the digital Retrospective board served a function that would probably be missed in common Retrospective Meetings. Such aggregated hypotheses were first possible to uncover when given the time and opportunity to look at all observations from a birds-eye perspective.

7.1.2. Lack of cross-functionality

Secondly, cross-functional development has been a subject of endless discussion. The importance of cross-functional teams, underlined by the focus on collective code ownership and pair-programming, is not only discussed in the academic literature, but also in the Agile Manifesto (Beck et al., 2001). In projects with a scope comparable to this, knowledge about coding-paradigms, specific technologies and development platforms all come into play when the product being developed is, technology-wise, multi-faceted. The Agile Manifesto promotes collective code ownership to circumscribe *information silos*[11], making sure that all developers are knowledgeable about every part of the system in question.

Cross-functionality is an idea that is great in theory, but very hard to ensure in a development team with limited resources and substantial time-constraints. The amount of technologies and coding-languages involved (consider section 3.2), as well as the level of knowledge needed to be able to produce worthwhile code, implied that cross-functionality was hard to implement team-wide. Our inability to adequately address this issue was highlighted by observations 4, 5 and 16. We also see that this category overlaps with *Lack of technical competency* from the waste categories described in section 2.2.2.

Instead of trying to make each developer a Jack of all trades, the team had previously decided on a simplification of the cross-functionality principle. Each technology, and thereby each part of the system, was designated a lead and a secondary developer. This enabled some level of pair programming on complex problems, and relieved the burden of aggressive learning on most of the team. Instead, the developers could understand more thoroughly their designated technology,

[11] Denoting an entity, often a person or an actual system, having unique information or knowledge not shared with or by the rest of the ecosystem.

and as such create better code more efficiently. However, as the observations highlighted, the implementation of this principle was not without its flaws.

When looking at the literature, we see that Ikonen et al. (2010) mentions *waiting* as a waste category, and specifically waiting when a person requires assistance. The observations that were reported in this category seem to have resulted in having to wait for the lead developer when experiencing, for instance, faulty code, or an API-function that was not implemented in the expected manner. Oftentimes, there was a discrepancy between how different developers understood the underlying requirements of a function. This made the testing between different platforms harder, and the dependencies between the data-driven web-application and the REST-API became obvious. Another contributing factor that was mentioned was that some developers committed new versions of the program and uploaded their version to the project Git-repository without sufficient testing. This also made the development of new modules that were dependent on others more cumbersome, as the REST-communication broke down in unexpected places and situations, and often different from version to version when coding progressed. Here, lack of communication came into play, a category also cited by Ikonen et al. (2010).

7.1.3. Lack of clear prioritization

Lack of clear prioritization became a problem in need of remedy. As have already been explained, long-term planning was not an option for our project, as many of the development tasks became contingent on whether new system integrations with third-parties was to be performed. During this initial data-gathering period, the company was having multiple discussions about service integration opportunities, with other established product lines. As often is the case in the business world, such discussions tend to take quite some time, and the team was unable to complete the discussions with any of the potential integration partners in the initial period. This created a vacuum in the subsequent prioritization.

If one of the integration-agreements were struck, the most rewarding revenue stream would have come from the direct link with other third-parties and their respective customers, meaning a complete shift of focus would be inevitable. In this case, the most pressing tasks would stem from the mobile application, and not the web-application. If the agreements did not come through, most of the remaining work would be in the web-application, since it would provide us with the most credible revenue stream. This conundrum was hard to treat in a specific manner, because of the initial enthusiasm from the third-party companies involved in the discussion. However, the conversations went slow, leaving the development team in a state of not knowing where and how to best spend their development time.

Here, we see an example of an anticipated process *in vivo* (see section 2.4.6), and the development team found themselves in a situation with seemingly no correct next move. As I argued in section 2.4.6, the development team was wasteful in their inability to decide on what to prioritize, until additional information would force a different route to be taken.

We can also see that this category loosely overlaps with *lack of control,* a category mentioned in 2.2.2. However, we could make a case that this waste-object is more specific in character.

7.1.4. Lack of follow-up of approved proposals for improvements

The last category of the 1st iteration was concerned with our inability to see proposed changes through. These observations (7, 8, 9 and 14) were concerned with our plan of hosting a Daily Stand-Up Meeting at the start of one of the weekly meeting days. However, one or more of the team members were often running late, because of, for instance, misunderstandings, sleeping too long or sick children. This became an annoyance for quite a few participants, where nothing of value was produced as we waited to start the meeting. To circumscribe this issue, we tried to stress the importance of showing up at the specified time. Generally, in these discussions temper often ran high. This did not result in any visible improvement of the issue.

Seeing that our previously proposed solution did not improve the issue at hand, we decided to change the routine entirely. As a response to the issue, we decided that instead of having the meeting at the start of the day, we chose to host the meeting as the last task we did during the co-located sessions.

This finding could be explained as a case-specific conglomerate combination of *waiting, inadequate corporate culture* and *unwillingness to train and learn.* However, none of these categories were sufficient in solely explaining the registered observations, and it could therefore be classified as a new category.

7.1.5. Minimally sufficient explanation

All the above observations and insights were obtained without the need of the Process Tracing analysis, i.e. without interpreting the observational data in a sequential manner. By looking at the data set, these four categories were obvious, and were covered to a certain extent by the waste categories and critical success factors literature. Therefore, one could form a *minimally sufficient explanation* (MSE) (see section 2.4.1.3) of the waste-objects residing in the project, by taking the conjunction of these four categories without usage of the tests described in section 2.4.3. This MSE covered twelve of the sixteen observations made by the development team members. Two of the remaining observations could be deemed specific for this project, as they related to a specific integration opportunity.

The minimally sufficient explanation of the waste-objects was therefore described as the following three points:

MSE-I: *We are unable to compose a unified perception of what tasks to prioritize, both business and development-wise, mostly due to our potential integrations.*

MSE-II: *Too little focus on collective code-ownership, leads to developers having to wait for assistance, because of their lack of knowledge about the problem domain, or coding paradigm, at hand.*

MSE-III: *Lack of respect for the collective activities, such as arriving late at meetings, results in pushbacks and inefficiency.*

7.1.6. Retrospective Meeting

By using this conglomerate MSE as our meeting agenda, we increased our efficiency and ability to propose actions to eliminate the described waste-objects. Having such an agenda turned our meetings from being primarily explanation-oriented to a problem-solving exercise, as the participating team members had had some time to reflect upon the waste-objects prior to the meeting. Having a focused agenda prior to the meeting enabled us to cut 45 minutes of the assigned time (1h 15 min instead of the previous time-box of two hours), while at the same time proposing actions that could, grounded in observations, help increase our development efficiency. In relation to the MSE, these actions were:

ACT-I: *Keep strategic discussions to the newly established strategy meeting.*

ACT-II: *Assign more time to develop our cross-functional coding skills, promoting pair-programming between both backend and frontend lead and secondary developers.*

ACT-III: *Set the Daily Stand-Up Meeting as the last event of the co-located coding sessions.*

7.1.7. General findings

As such, the analytical framework described by Van Evera (1997) and discussed in section 2.4.2 seemed, initially, to be redundant and wasteful, as a well-founded minimally sufficient explanation was obtained without a thorough, causal analysis. As a technique for identifying waste, it is necessary for Process Tracing to be efficient, and spending hours conducting a redundant analysis would, in itself, be wasteful.

These findings are interesting when discussing the nature of waste in Agile and Lean development projects. What we can take with us is that, at least in this iteration, waste-objects seemed not to be bound *temporally* and *causally*, but *categorically*. In the analysis of the waste-objects, the only hypothesis that was best described in a causal manner was MSE-I, where uncertainty about possible business-opportunities lead to the inability of weighing development tasks against each other, and therefore wasteful usage of time. MSE-II and III were better explained categorically, by which I mean that they had no prime mover. They did not represent the first node in the tree of wasteful *events*, but rather a collective set of *attitudes* that needed to be altered. This is also an argument against using Process Tracing as an analytical framework *in this setting*, as it relies mainly on the principle that events happen in a temporal order, and that

there is a unique start of the causal chain that lead to the specific result. This seems not to be the case with waste-objects and their subsequent identification. It might be the case that this result came by because of the parameters of this study, which included a rather inexperienced team of developers with a project that had been running for only one year at the time of analysis.

In this iteration, a full-scale Process Tracing analysis was neither necessary nor meaningful. However, using the specified data gathering technique seemed as an effective way of storing data in a temporal order, which also enabled an easy subsequent categorization. In teams that have a longer track-record, composed by more experienced developers, it might be the case that waste is not best described in a categorical manner, but that single events have a bigger impact on the coding efficiency than team member behaviour seemed to have in this project.

7.1.8. Improvements to the technique

Having completed the first iteration, we deemed the initial implementation of the technique a success. Having produced an MSE of the waste-objects prior to the Retrospective Meeting and using this as an agenda, enhanced the meeting's quality; a sentiment shared by all participants of the study. The only improvement to the technique, that was evident following the analysis, was to also store at which time the participants had experienced their observation, and not just its date. This would help the researcher in finding more fine-grained patterns and relations. For instance, an observation stating that other developers were not available would not be of equal importance if it was reported at 03:00 in the morning than, say, if it was reported within the designated co-development coding sessions. For the next iteration, the time of observation would also be included in the observation reporting scheme.

7.2. Iteration II

The following four categories were found in the 2nd iteration:

1. Substantial time-constraints
2. Lack of motivation
3. Third-party collaboration
4. Poor coding standards

In this analysis, we found less reports citing observations of the categories defined in the 1st iteration; indicating that we had been successful in addressing and solving issues specifically relating to the categories *lack of cross-functionality* and *lack of clear prioritization.* This means that the list of actions for improvements constructed in the last Retrospective Meeting had, at least partially, increased our ability to respond to the project's waste. However, we still experienced observations tangential to the categories *lack of planning* and *lack of follow-up of approved proposals for improvement.*

The latter two, as previously addressed, did have some shared observations. This meant that they had a closer coupling than the rest of the observation set and its assigned categories. The tangential reports were in this iteration classified as *lack of motivation*, as the reports, albeit quite similar, cited different situations than in the preceding iteration.

7.2.1. Substantial time-constraints

Secondary Frontend and Backend both pointed to the fact that when working as seldom as we did (two days of co-located development each Tuesday and Thursday), we effectively created an inefficient structure for those having to deal with multiple technologies. These two developers were not lead in their fields, but secondary developers mainly tasked with supporting Lead Frontend and Backend. Having to jump between tasks and code-bases made it hard for them to contribute immediately, since they had to spend a considerable amount of time to simply understand the structure of the system they worked on, from session to session.

We interpreted this as a result of the lead developers actively producing code between the co-located coding sessions. In addition, we also suffered from a lack of resources, both in terms of assigned working hours and actual system developers. However, it was a situation that we had to deal with in some way, and it became clear that we should choose a different route when structuring our working habits. By looking at the waste categories described in section 2.2.2, we can define this category as a case-specific instance of Ikonen et al. (2010)'s category *task switching*. As such, a case can be made that this category has been discussed in the waste literature previously.

Time constraints and lack of resources are of course not directly *waste,* but pointed, in this instance, to a case where our co-located processes were not optimized to cope with our current team structure. Here, we saw a possibility to reschedule our co-located sessions to better spend the time of the secondary developers. This was discussed in the subsequent Retrospective Meeting.

In other development teams, such a lack of resources might have been solved by assigning more developers to the specific tasks that were suffering, or by granting more work hours to those already involved with the coding. We, however, did not have the opportunity to do this, making a rearranging of our co-located coding sessions the only feasible option.

7.2.2. Lack of motivation

We found that some of the same types of observations, like those previously coined *lack of planning* and *lack of follow-through of proposed improvements,* were still reported. For instance, Lead Backend stated that "*people* [were] *absent on short notice",* which he continued to reiterate in the observation reports two more times. It became quite clear that this issue related to specific individuals, and that our plan of moving the Daily Stand-Up Meeting to the end of the day had

served its purpose for all other types of previous observations. This seeming lack of motivation from certain individuals was addressed prior to the next Retrospective Meeting.

Considering the literature, we can see that there are multiple sources citing the importance of certain personal characteristics. According to Matook et al. (2014), "*You need personal discipline in order to comply with the few rules that Agile ISD methods such as Scrum are opposing.*" In addition, their findings also seemed to imply that "*Only motivated individuals enable efficient teamwork in an Agile project*". Therefore, we can see that this category has been discussed in the literature previously.

This statement held true with the participating team. Often, we had agreed to complete specific tasks in advance of the co-located coding sessions, as to increase our total hours spent on complex problems that one needed knowledge about multiple technologies to be able to address appropriately. We experienced that some, who had been tasked to produce specific functions or modules, neglected their responsibility, and as a result deemed it more practical to work from home. This was of course not acceptable and was considered a critical point to address.

7.2.3. Third-party collaboration

We still lacked proper documentation from the third-party software development firms we were in contact with, but had midway through the development period been able to circumscribe most of the issues. This had hindered effective coding towards their API's, but was somewhat resolved when we got to grips with our initial problems. This type of exogenous cause was hard to deal with in any specific manner, as it was a unique opportunity for us to nevertheless produce this functionality, from a strategic business perspective.

Looking at the literature, we cannot find any instances of this type of waste directly. Many papers, such as Bermejo et al. (2014), Darwish et al. (2015) and Nord et al. (2013), do however mention both *customer collaboration* and *commitment* as critical success factors. Yet, these are broad categories in nature and do not provide any sensible feedback in how to resolve such case-specific instances once they arise.

7.2.4. Poor coding standards

A lack of collectively defined coding standards became a frustration for the backend-crew when multiple developers participated on the same module. This became obvious when Lead Backend had to rewrite a part of the system to fit a new database-communication protocol. Having a shared "vocabulary" when coding would have served to limit misunderstandings when taking over the code of others, and this should have been stressed more thoroughly before introducing new members to an existing code-base. This also highlighted the fact that we had not utilized procedures such as pair programming adequately, meaning that some level of individual habits made a mark on parts of the code. One explanation of this occurrence might also be that different

developers primarily worked with different coding languages. For instance, C# and TypeScript do share similarities, but their *best practices* might differ substantially.

The waste literature does not cover this issue, to the best of my knowledge. Neither does the literature involved with critical success factors, at least at such a detailed level. However, many sources mention both *control* and *corporate culture* as important success factors in establishing an Agile development project, i.e. Stankovic et al. (2013) and Nord et al. (2013). In addition, we can see that *inadequate technical competency* was mentioned in the waste categories defined in section 2.2.2. However, this instance did not seem to relate to the skill-set of developers, but rather to the fact that specific team members had not worked much together previously. Therefore, these members had not had the opportunity to agree on a specific set of design patterns and coding standards for each of the code-bases.

On the other hand, if we look outside the scope of critical success factors and waste categories, we can see that one of the core principles of eXtreme Programming is that "*Code must be written to agreed standards*" (Wells, 1999). For most software engineers, this might be obvious, but project managers might nonetheless overlook such sources when looking for information regarding process improvement.

7.2.5. Minimally sufficient explanation

By understanding the categories cited above, the researcher could formulate the minimally sufficient explanation as follows:

MSE-I: *Having co-located coding sessions only Tuesdays and Thursdays implies that the secondary developers must spend a considerable amount of time getting to grips with the code base from session to session, as the lead developers produce code between these sessions.*

MSE-II: *There seems to be a lack of motivation from certain individuals in our development team, which leads to pushbacks of some modules.*

MSE-III: *Poor coding standards contribute to misunderstandings and frustrations of those having to refactor or reformulate some functions.*

MSE -IV: *Lack of feedback from the third-party service providers makes our integration with these providers inefficient.*

7.2.6. Retrospective Meeting

This conglomerate MSE was again used as our agenda for the Retrospective Meeting, which was deemed a success in the 1st iteration. This meeting only lasted one hour in total, and no other points of discussion was put forward by the team after having explored the four points of the

agenda. After discussing each of these points, the team agreed upon a list of actionable proposals for further improvement, which included:

ACT-I: *Instead of having co-located working sessions two full days each week, we should work four full days every 2nd week. This should serve to reduce the total amount of time spent on understanding the code base from session to session.*

ACT-II: *Specify a list of coding standards relating to structure, naming conventions, etc. This should be discussed further in the strategy meeting.*

ACT-III: *Define a "key account manager" tasked with getting the necessary feedback and documentation from any new third-party service providers.*

MSE-II was of a more sensitive character, relating to a certain individual's lack of motivation to produce the necessary functionality. This was a matter best addressed in solitude with that individual ahead of the meeting, and it seemed as if our suspicions were correct. One of the members said he had struggled to find the internal drive to pursue the start-up path, instead of applying for a paying job. This was of course understandable, and he decided to back out of the company, transferring his shares to the remaining owners. This meant that his development tasks had to be transferred to other developers, and that the team had to settle for a smaller project scope than what was originally planned.

7.2.7. General findings

The 2nd iteration lent further evidence against defining waste-objects in relation to its sequential position in the data set. Instead, as with the 1st iteration, we found that the waste-objects could easily be understood from the perspective of categories, and that most of them did not have any "triggering event" that started a chain of wasteful behaviour. If such categories lead to actionable proposals for improvements, as they so far had done, there should be no need of using the four tests described in section 2.4.3. Performing such a procedure could be costly in terms of financial resources, and would require more time than a categorisation. The categorisation took, in this iteration, 30 minutes to perform, which was sufficient in establishing the Retrospective Meeting agenda that was utilized. This agenda was also apparently sufficient in covering most aspects of the traditional Retrospective Meetings, as it:

1. Could inspect how the last Sprint went with regards to people, relationships, processes and tools;
2. Identified and ordered potential improvements; and
3. Enabled the creation of a plan to implement those improvements.

At an abstract level, these aspects covered the basic purpose of the Sprint Retrospective in its entirety, while at the same time ensuring that the meeting was held in an efficient manner.

7.2.8. Improvements to the technique

In the 2nd iteration, the data gathering scheme was expanded to also include the time of observations. This was not a necessary trait, considering the categorization described above. Yet, it did provide valuable background information about the observations, at a very low cost. As such, we decided to keep this trait also for the 3rd iteration.

Essentially, there were not found any negative aspects with the technique. By the second iteration, we had seen that the data gathering technique was successful in adequately defining our case-specific waste-objects, to such an extent as to enable rapid, and seemingly correct, categorization, that led to actionable proposals for improvements. Therefore, the technique was kept as it was for the last iteration.

7.3. Iteration III

The last iteration uncovered the following three categories:

1. Substantial time-constraints
2. Poor technical solution
3. Poor distribution of resources

During the 3rd iteration, the team did not register any observations that had been described in previous iterations, with exception of *substantial time-constraints* and, to some extent, *lack of cross-functionality*. The former was a recurring theme, and did become more prominent when entering the final phase of the participants own master's theses. The latter had manifested itself in diverse ways during the three iterations, and we had taken measures to combat the lack of cross-functionality already. The data set seemed to imply that we had been able to create actionable proposals for improvements for most previously stated MSE's, as well as being able to implement said proposals in an efficient manner.

7.3.1. Substantial time-constraints

As was also addressed in the previous iteration, some team members were experiencing an acute lack of time to be able to effectively produce the necessary code within the agreed-upon timeframe. However, this was not related to the project structure no longer, but rather to the fact that exogenous intervening factors had come into play. Mainly, this related to the participants need of focusing on their own master's theses, and as such did not have as much time to allocate to the company as previously. We knew this would eventually become an issue, and there simply was no way to circumscribe this in the remaining months before their theses' delivery. As such, the team experienced a decline in development pace, which we attributed to this unfortunate, albeit necessary, shift in focus.

7.3.2. Poor technical solution

Before the start of the project, the team had decided to produce the frontend-specific parts of the software by usage of the framework named Angular. This framework was deemed the most interesting and provident solution for us at the time, and was expected to become the new "industry standard" within large-scale frontend-development. However, the framework was updated more than frequently, leading to problems with versioning and dependencies. As the framework was dependent on many other systems (*which often had to be exempt from the repository and installed directly on the developers' computers*), such as Node, this became a hassle when involving multiple members on the development of certain modules.

Here, we saw an acute lack of knowledge about how to solve such issues relating to files not submitted to the git-repositories, as well as ensuring that the right versions of both Node and Angular-CLI was installed and utilized by all the participating developers at the same time. This led to set-backs and inefficient development, where team members at times spent as much as one hour updating their software before the coding could commence.

These types of problems are not mentioned as waste by Ikonen et al. (2010), and neither can we find any direct references to such problems in the critical success factor literature. The closest category stated in section 2.2.2 is *inadequate technical competency,* but this category was not directly applicable in this case, as it had nothing to with development skills, but rather to the fact that centralized versioning and dependency-handling was not a possibility. Node, for instance, is installed directly on each of the developers' own computers, and could not be submitted to our git-repository[12]. Such issues might have arisen lately since large-scale frontend development has, in recent times, become more aligned with traditional OOP-systems in terms of complexity [13].

7.3.3. Poor distribution of resources

In addition to the dependency-issues we faced, we also saw that having an acute focus on integration-development lent little time to produce the necessary API-methods needed to support continuous frontend-development. By this time, we had a full list of new functional requirements that should be implemented, but did not have the time to produce both integration-functionality and the methods supporting the web-app. This resulted in an ineffective development of the frontend solution.

In this case, the category *poor distribution of resources* could be explained by the category called *waiting* observed by Ikonen et al. (2010). Explicitly, they mentioned that *"a person occupied when someone requires assistance"* should be considered waste, and this seems like a good example of this description.

[12] A versioning-control software: See www.bitbucket.com for additional information.
[13] Salsita (2015) provides a good discussion about this segment.

7.3.4. Minimally sufficient explanation

To explain these categories, the following MSE was formed:

MSE-I: *Dependency issues pertaining to different versions of needed software between the developers' computers leads us to spent a lot of time just updating the software when we should develop new functionality co-located.*

MSE-II: *Having the backend-crew only developing the integration functionality leads the frontend developers to rely their development on test data, often created ad hoc.*

The category *substantial time-constraints* was not addressed, as I saw no point in addressing the fact that the participants had additional responsibilities outside the project, especially so when they were in the middle of finishing their own theses. This would only serve to induce more stress.

7.3.5. Retrospective Meeting

The agenda of the Retrospective Meeting was again composed by the minimally sufficient explanation cited in the previous section. As we could see, these issues related mostly to the frontend developers, and the team's high-level prioritization of tasks. The developers agreed that we should first and foremost focus on the integrations, rather than the web-application. As such, we agreed that now would be a good point to enhance the frontend developers' cross-functionality, and therefore decided that Lead and Secondary Frontend should also be tasked with work relating to these integrations.

However, we still needed to address the dependency issues that we had encountered, as we would have to deal with these systems sooner or later. As such, we decided that updating the systems on all the involved developers' computers should happen on an agreed-upon date in advance of each Sprint, ensuring that all involved developers had the same versions of all the software used, throughout the development period. This should serve to limit both dependency-issues between different libraries, and should also serve to limit the concurrency-issues between the developers work stations. Therefore, we established the following list of actionable proposals for improvement:

ACT-I: *Update the development software (Node, Angular-CLI and Angular) concurrently in advance of new Sprints.*

ACT-II: *For now, have the frontend developers enhance their backend coding skills.*

Devising this plan took in total 30 minutes, and no other discussion were had relating to waste.

7.3.6. General findings

Generally, there was not much additional information to be gained about either the technique or waste during this sprint. We had seen a decline in development pace because of the developers' other responsibilities, and had not been able to prioritize the product development as highly as with the other Action Research cycles.

However, we could form the same conclusion as previously, namely that waste seemed to be understood more easily by conducting a categorisation, rather than a sequential analysis trying to make out triggering events in the data set. This does of course have a lot to do with the way the data material was produced, and it might have been that conducting interviews during the cycles could serve to bring forth more clues about such sequentially-defined mechanisms. Yet, one main goal of the study was that it should produce an organizational artefact that was efficient, and therefore the categorisation was deemed the most cost-effective technique available.

7.3.7. Improvements to the technique

No aspects of the technique were deemed sub-par, as the actionable proposals for improvement from the last iteration were considered successfully acted upon. The level of detail of each observation was also seen as sufficient to understand the situation, time and type of the underlying waste-objects. In addition, there had been no information pointing to the technique underperforming in relation to this iteration, and the analysis had been successful in addressing the most pressing concerns of the participating individuals.

As such, we had established a technique that was deemed stable in criteria. The technique had also been successful in informing the participants about the waste residing in the project, which furthermore enabled the team to produce actionable proposals for improvement.

8. Results

This chapter will be segmented into two parts; one detailing the complete description of Retrospective Process Tracing, while the other serves to answer the research questions postulated in the introduction. This will enable the reader to get a high-level understanding of Retrospective Process Tracing, as well as an understanding of the extent to which findings can be generalised and, therefore, answered.

8.1. The Technique

The three iterations terminated with a descriptive technique that had shown to consistently result in actionable proposals for improvement, helping the team reduce waste in their project.

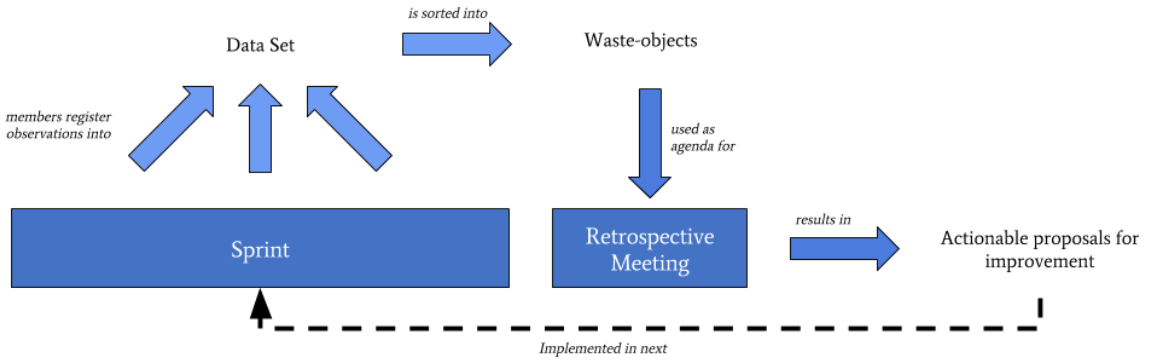

FIGURE 11: GRAPHIC VISUALIZATION OF RETROSPECTIVE PROCESS TRACING.

The technique, as visualized above, starts off on the left-hand side. Here, the team members register observations throughout the development period into a data set. At the end of the Sprint(s), but before the Retrospective Meeting, these observations are categorized into descriptive waste-objects explaining the provided observations.

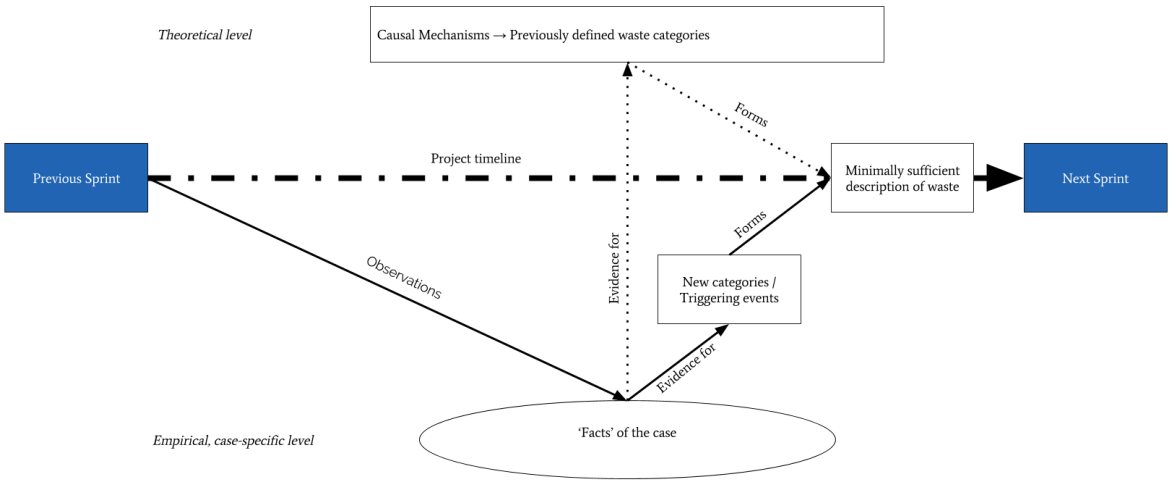

FIGURE 12: GRAPHIC VISUALIZATION OF THE MANIPULATED EOPT CATEGORIZATION TECHNIQUE.

These waste-objects are either supportive of previously defined causal mechanisms, lend evidence for new, case-specific causal mechanisms, or implies that a specific triggering event had caused inappropriate handling of the situation to ensue. After such waste-objects have been formed, they are sent to the development team members as an agenda for the subsequent Retrospective Meeting.

The goal of the Retrospective Meeting is then primarily aimed at providing actionable proposals for eliminating these waste-objects. As such, the Retrospective Meeting results in a specific plan to eliminate all waste-objects that have been found. This plan is then acted out in the subsequent Sprint, which will serve to reduce the observed waste-objects, and the process starts over.

8.1.1. Additional concepts

Retrospective Process Tracing consists of three new high-level concepts not described in the Scrum Guide. These include:

1. Waste observations
2. Waste-objects
3. Actionable proposals for improvement

8.1.1.1. Waste observations

The technique relies on participant-registered observations that signify possible waste. These observations are registered in a digital scheme, which include the following parameters:

1. **Observation ID:** *Incrementally assigned number of observation.*
2. **Date of observation:** *The date that the developer experiences their observation.*

3. **Time of observation:** *The time that the developer experiences their observation.*
4. **Participant ID:** *A predefined ID of the developer citing the observation.*
5. **Description of observation:** *A description clarifying what was considered waste.*
6. **Situation (*optional*):** *Some waste-objects can be furthermore explained by also describing the situation that the developers find themselves in when observing the waste.*
7. **Potential actions (*optional*):** *If the developers observing the waste have any immediate thoughts about how to fix the issue at hand, they may choose to provide them.*

This set of parameters was shown to enable an efficient categorization. The only parameter that was added during the Action Research phase was *time of observation*, which could serve to provide additional background-information about the underlying waste-object. This parameter was not seen as having an important effect on the analysis, but providing this parameter did not add any overhead worthy of discussion. Therefore, the final set of parameters is the one cited above.

8.1.1.2. Waste-objects

The waste-objects are condensed representations of the waste observations. These were, in the study, created by analysing the waste observations and looking for patterns, or observations that seemed to be connected either by time, type, or the situation they incurred in. The observations were mapped from the data set to a respective waste-object or triggering event, new or previously defined, solely by the researcher, and should be an easy job for any project manager to perform. These waste-objects thus represent the state of waste in the project, and are taken as an agenda for the next Retrospective Meeting.

8.1.1.3. Actionable proposals for improvement

With Retrospective Process Tracing, the Retrospective Meeting's purpose is to address the waste-objects that have been identified during the Sprint. As such, it is in the Retrospective Meeting that a plan for eliminating these waste-objects is agreed upon. Each of the waste-objects should be addressed, and the development team should create a list of *actionable proposals for improvement* to eliminate each of these instances. This list represents the complete set of changes, additions or eliminations to processes, events or behaviour that should be instigated during the next Sprint.

8.2. Research Questions

The study enabled me to conclude on all the research questions, but with differing levels of both confidence and generalizability. These will be detailed in the following sub-section.

Research Question I: *Is it possible to identify waste-objects not covered in the existing waste literature by using Retrospective Process Tracing?*

There were numerous waste observations and waste-objects that lent evidence towards concluding that it was, in fact, possible to identify types of waste not covered in the existing waste literature by using this technique. Iterations I and II highlighted specific instances of waste that were subject to the parameters of the participating team, and that, provided a birds-eye perspective, we could understand how these observations related to the rest of the data set. The waste-objects that were found in this project, and not in the waste literature, are summarized below:

- *Lack of follow-up of approved proposals for improvements*	Section 7.1.4
- *Third-party collaboration*	Section 7.2.3
- *Poor coding standards*	Section 7.2.4
- *Poor technical solution*	Section 7.3.2

The exploratory basis of this study does, however, imply that external generalizability cannot directly be guaranteed. Here, parameters such as team size, participant motivation, organization, distribution of developers, and so on, all have implications. Therefore, we can conclude that Retrospective Process Tracing enabled the *participating team* in uncovering waste-objects not described in the literature, but that any further generalization does require verification studies.

Research Question II: *Can Retrospective Process Tracing be implemented with a relatively low initial cost?*

By looking at each of the iterations, we can conclude that the technique in effect saved resources. In iteration I, II and III, we saw that the Retrospective Meeting took 1h 15min, 1h and 30min, respectively. Previously, the team had experienced problems with schedule overruns, and this was no longer an issue. In addition, the team members reported that registering observations usually took no longer than three to five minutes.

Considering these numbers, we can see that, with sixteen observations, the developers spent in total 80 minutes (16 x 5 = 80) registering observations. The researcher spent between 30 and 45 minutes categorizing the observations and abstracting the minimally sufficient explanations. This results in a spending of resources totalling 125 minutes on observation reporting and analysis.

With the Retrospective Meeting requiring maximally 75 minutes of the collective teams' time, the cost of the actual meeting was 75 minutes x 6 developers participating = 450 minutes. This makes the total cost of the technique 575 minutes, or approximately 9,5 work hours. Additionally, the Retrospective meetings were only held every 2nd two-week iteration, which meant that the actual cost would be 575 / 2 = 290 minutes, per iteration. Here I have not factored in the time spent on the initial educational phase or the problem identification phase, as I would argue that these were research-dependent, and not required for using the technique, once it had been produced.

If we also consider the new strategy meeting that had been established during the problem identification phase, the calculations become a bit more obscured. However, these usually took less than one hour in total, and were held each iteration. This implies a cost of 360 minutes, which totals at 650 minutes, or approximately 11 hours, for both Retrospective Process Tracing and the newly formed strategy meeting.

By comparison, a traditional Retrospective Meeting time-boxed for two hours, with 6 participating developers would imply a total cost of resources of 720 minutes, or 12 hours of work. This means that, within the confinement of this team, the collective cost of both Retrospective Process Tracing and the strategy meeting were lower than that of traditional Retrospective Meetings, enabling the researcher to conclude that the technique in effect saved the development team working hours.

Research Question III: *Is Process Tracing enough to serve as a substitute to traditional Retrospective Meetings, or should it be used as a supplement?*

To answer this question, we must take a look back at the Scrum Guide. Here it is stated that the goal of a Sprint Retrospective is to:

1. Inspect how the last Sprint went with regards to people, relationships, processes and tools;
2. Identify and order the major items that went well and potential improvements, and;
3. Create a plan for implementing improvements to the way the Scrum Team does its work.

Retrospective Process Tracing, as should be evident from the data analysis, solves at least points 1 and 3. The technique enabled an efficient inspection of the Sprints, primarily with regard to processes and tools, but it also helped to uncover problematic situations with regard to *people and relationships* (see category *lack of motivation, section 7.2.2*). The technique also enabled the creation of an evidence-based plan for improvement, as a response to the described MSE's of waste, which the team were largely successful in acting upon.

Regarding point 2, there is room for a bigger discussion. The technique does identify potential improvements, but does not provide any specific ordering of these. The explicit necessity of this might be possible to assess in further studies, but the team was, as stated, largely successful in acting upon all potential improvements that was uncovered.

The technique also does not identify "*...the major items that went well"*, which might be a bigger setback. One argument in favour of traditional Retrospective Meetings is that it can function as a motivational boost for the development team, serving as collective "pat on the back". The motivational aspect of Retrospective Meetings has not been a subject of discussion in this thesis, and could also be possible to assess in further studies.

Therefore, we can conclude that Retrospective Process Tracing can, to the extent of this team, serve as a substitute to the traditional Retrospective Meetings. The technique accomplished two out of the three ascribed goals, while the last goal was partially accomplished. Using the technique also reduced the participants' need to discuss any frustrations and concerns, as they were already covered by the registered observations, and were factored into the minimally sufficient explanations.

The necessity of Retrospective Meetings to serve as motivational boosters was also not needed in this study, where the team members were mostly interested in increasing waste identification capacity, and had a strong internal motivation to succeed with the project (with one exemption, as stated in sections 7.2.2 and 7.2.6). This might be different in other teams, with members having different personal traits from that of the participants.

9. Discussion

This study aimed at adapting Process Tracing to the field of waste identification within Agile and Lean development teams. To achieve this, the researcher chose to perform an exploratory Action Research study with a team consisting of six developers over a three-month period. The study terminated with a new technique, stable in criteria, that was deemed better at establishing actionable proposals for improvements, compared to that of traditional Retrospective Meetings.

This chapter will highlight the most important findings concerning the concept of waste in general, and more specifically concerning the identification of it. These findings will be discussed in relation to relevant literature, what it means for the industry, and how they can contribute to the field of software development research. In addition, the chapter will terminate with a self-assessment of the study in relation to the standard qualitative evaluation criteria for validity: *Credibility*, *transferability*, *dependability* and *confirmability*.

9.1. On the Nature of Waste

The three Action Research cycles showcased an attribute that waste-objects seemingly embodied within this team: Namely, that observations were bound primarily *categorically*, and not *sequentially*, as first hypothesized. One possible explanation for this is that it was a result of the primary data-gathering technique that was utilized. If a more thorough data-gathering routine had been instantiated, sequential patterns might have emerged. However, this would make Retrospective Process Tracing less cost-efficient, and was therefore neglected as a viable solution.

The study failed in finding (internal) triggering events that lead to wasteful behaviour, but were nevertheless able to produce the necessary descriptions for waste to be eliminated. This might be a result of the participating team, but at the time of writing, there is no viable way to substantiate or undermine this claim.

The categoricity of the waste observations, although case-specific, does bear importance for the waste literature. Primarily because it is the usual way success factors and waste descriptions are defined, such as section 2.2.2 has sought to highlight. This means that sources of knowledge about waste and success factors have a compelling case, and should be deemed important resources for project managers who might seek to understand waste in their own development teams.

The descriptiveness of the technique was also a positive trait. By using the technique, we were not only able to find instances of previously hypothesized waste categories, but it also enabled the formulation of well-defined descriptions of new, case-specific waste-objects. In hindsight, this descriptiveness might be more important for the technique than its capacity for identifying waste categories.

An investigation of the open-channel communications between participants did not result in any useful pieces of observations. However, it did help in understanding the context of some observations, and provided evidence that the developers who had recorded observations often formulated their concerns to other participants. Therefore, ensuring some form of motivational primer for reporting observations seems both necessary and valuable for the result, as it is the registered observations that are the prime drivers to establish an effective Retrospective Process Tracing analysis.

Additionally, the study uncovered multiple sources of waste relating to anticipated processes. These processes, events that might happen outside the scope and control of the team, made prioritization of engineering-tasks difficult. This serves to highlight the importance of contingency-plans when dealing with multiple stakeholders.

Ensuring that everyone knows what to do, in all cases, was deemed a necessity after uncovering our failure to adapt to new, not-planned-for situations. This feature of waste has been discussed, to some extent, considering typical Agile traits such as “change management skills” (Ahimbisibwe et al., 2015), but exactly what such a skill-set amounts to is still quite an open question. The impact of anticipated processes on a development team’s ability to prioritize work is an area that might deserve additional attention.

9.2. On Retrospective Meetings

By studying the iterations, we can see that the goal of the Retrospective Meetings changed focus. By using Retrospective Process Tracing, the members were enabled to host a solution-oriented event, primarily interested in establishing a plan of attack, considering the inefficiencies within the project. This feature of Retrospective Process Tracing is interesting, as it seemingly increased the team members’ awareness of wasteful behaviour, and therefore inherently made them more interested in understanding their own habits. The Retrospective Meetings’ change in focus was enabled by the data gathering technique used within the study. Being able to present a list of areas to address prior to the meeting also served to limit irrelevant discussions.

Having such a data set also showcased the importance of understanding observations from a birds-eye perspective. By being able to look at specific observations in light of other observations, we could instantaneously probe which areas had been reported with a high frequency, how they were distributed, and whether they had been recurring points of frustration for some members during the preceding development period. This enabled the researcher to get a glimpse of what the *real* cause for the observations were, and thereby being able to provide an accurate description of the underlying waste-object. This specificity is lost on the waste categories described in section 2.2.2.

9.3. Iterative & Continuous Development

In this study, Retrospective Process Tracing was used within the confinement of iteratively organised events. However, it does not explicitly *rely* on such a structure, and should be possible to use within continuous development settings such as Kanban and DevOps. The only point to keep in mind is that the technique does require a bare minimum of observations for it to have any sensible purpose. This means that, considering continuous development, the Retrospective Meetings could be held *ad hoc* when the team have registered enough observations to form a minimally sufficient explanation, enabling actionable proposals for improvement to be formed.

In some sense, it would be possible to argue that Retrospective Process Tracing should be a better fit for continuous development than iterative process structures. For instance, if the observational data set was small in numbers, the Retrospective Meetings would be redundant in Scrum. This was highlighted by our choice of hosting the Retrospective Meeting only every 2nd Sprint, as opposed to the traditional procedure of hosting the meeting at the closure of each Sprint. This would not be an issue if the meeting was planned *after* a minimally sufficient explanation of waste had been formed, at the leisure of the project manager, team leader or Scrum Master.

9.4. Weaknesses of the Technique

The biggest setback of Retrospective Process Tracing, as seen by the researcher, is that it relies heavily on the participants' internal motivation to produce observations. This is a feature that can be done very little about, and might serve as a critique against the technique.

How motivated a team is at improving their process of identifying waste, might be subject to many factors, such as place in organization, team size, distribution, level of top-down management and self-organization. Within the confinement of this team, it was not deemed an issue, but how well this translates to other groups of developers is a consideration that should be addressed in any further studies of Retrospective Process Tracing.

In addition, the technique relies on the "researcher's" (*i.e. project manager's, team leader's, or Scrum Master's*) ability to formalize waste-objects that are anchored in the observations. This is somewhat covered by looking at the set of waste categories first, to find observations substantiating their case-specific instance. If there are waste observations disagreeing with the waste categories in question, it might support the formation of case-specific waste-objects. Here, some level of analytical skill is unfortunately necessary.

9.5. Methodological Prospects

This paper represents a comparatively new way of conducting exploratory research in Information Science and Software Engineering studies: Firstly, it presents a new usage of the formerly underrepresented method Process Tracing, and secondly, it introduces Action Research

as a potentially viable method for improving upon methodological challenges, such as process optimization.

Both methods seem to have a high degree of usability in understanding concepts relating to Information Science and Software Engineering. However, Action Research goes one step further, in that it enables *practitioners* to improve upon their own processes, based on evidence and empiricism.

Collaboration between practitioners and academics entail a potential to address issues that neither can solve in solitude: Firstly, the insider-perspective might serve to highlight practical concerns, of importance to the practitioners, that traditional case studies might neglect to factor in. Secondly, it can serve as an anchoring-effect in *actual* practice, instead of *observed* practice.

This points to the potential benefits of using Action Research in exploratory studies within the context of software development practice and methodology. I would argue that it is important to question the accepted norms within frameworks such as Scrum, and therefore describe potential improvements to be made.

9.6. Evaluation

Conducting an exploratory Action Research study was deemed the only way to instigate a research method flexible enough to improve upon a previously hypothesized technique, realigning it with a new purpose. As a qualitative study, there were not all too many options possible to choose from, where traditional case studies were deemed the only alternative to Action Research. In hindsight, we can see that the three iterations did not instil any significant changes to the hypothesized technique, and a case study could therefore have been used instead. However, this was first possible to perceive post facto.

To evaluate this study, we can investigate to which degree the research design upholds the criteria of qualitative validity. These will be discussed below.

9.6.1. Credibility

Credibility refers to how well the research has established that the results are believable from the point of view of the participants. Trochim (2006) states that "*[...] the purpose of qualitative research is to describe or understand the phenomena of interest from the participant's eyes, the participants of the study are the only ones who can legitimately judge the credibility of the results*"

In an Action Research study, the results are *inherently* the views of the participants. This is furthermore established by the fact that the usage of Retrospective Process Tracing resulted in actionable proposals for improvement that the team was successful in acting upon. Therefore, the criteria of credibility should be met, since the resulting technique was based upon the input and

discussions held with the participants, transformed based on that input, and collectively deemed successful by the team.

9.6.2. Transferability

The criteria of transferability refer to how well, or to what degree, the results of a study are scalable, generalizable or transferrable to other situations with different parameters. By providing a sufficient audit trail, or enough information about why specific choices were made, we can enhance the "transfer"-value by enabling the reader to determine him or herself whether the findings are applicable in his or her situation. Therefore, some of the responsibility of considering the degree of transferability falls on those reading, or potentially citing, this master's thesis.

The explorative basis of the study does however imply severe limitations to any form of transferability. Here, we have seen a study involving six individuals, five of whom had known each other throughout their studies. They also had an acute interest in maximizing efficiency, which was deemed a direct return on investment. Therefore, a case could be made that the participating team do not translate well to other real-world development teams, situated in bigger organizations, where the developers might have lesser or different stakes, or different relations between themselves.

However, the same case could be made that the technique describes the situation that small-scale development start-ups usually finds themselves in. As a technique, it was successful in addressing the fact that the participants had other roles within the organization, having to perform work not directly implying value for the customer, whoever that might be. Whether this is a strength or a weakness should be up to the reader of this thesis to decide, based upon their own situation.

9.6.3. Dependability

To investigate the criteria of dependability, the evaluator checks to see if the researcher has been incautious, misread or misjudged evidence, or otherwise taken actions that could have led to mistakes and incorrect results. Self-assessment of this criterion is therefore impossible to perform with any merit.

9.6.4. Confirmability

The criteria of confirmability refer to what degree the results can be verified, corroborated or researched by others. In such exploratory studies, the actual findings *(i.e. waste categories)* are impossible to replicate. The research questions, on the other hand, do have such a character as to enable verification studies of Retrospective Process Tracing.

Hopefully, the study has presented enough contextual information about each waste-object that was uncovered, in addition to the actual waste observations, so that the same conclusions can be drawn from the data collection by other, uninvolved researchers. However, the master's thesis has not currently been corroborated by other researchers, or other development teams. Researching Retrospective Process Tracing with additional case studies might serve to verify these initial results. Such verification studies will have to be performed before any definite generalizability can be ensured for development teams in larger organizational structures, as well as for small-scale development start-ups.

9.7. Practical Implications

Retrospective Process Tracing, if verified by additional studies, could serve to transform the way companies and teams inspect themselves in relation to process-improvement and waste identification. It also stands as an example of why adhering to any specific framework is not always the best thing to do, promoting case-specific tailoring to Agile and Lean principles instead. As a technique, Retrospective Process Tracing might increase the participants' ability to both identify and eliminate waste, which could furthermore serve as a cost reduction for the team and company: Both in a human and financial way.

In addition, Retrospective Process Tracing enables the latter while not breaching the principles of either Scrum or Kanban. Therefore, it could be considered a waste identification technique that, potentially, have ramifications and is applicable for all teams adhering to the principles of Lean or Agile development.

10. Conclusion

This master's thesis has outlined an exploratory Action Research project adapting Process Tracing to a new field; the identification of waste within Agile and Lean software development. Over three month-long cycles I have shown the waste identification capacity of Retrospective Process Tracing, and how it helped transform simple participant-reported observations into actionable proposals for improvement. I have also outlined to what extent the team was able to implement said proposals, inviting to a bigger discussion about waste elimination, as well as laying out the total cost of resources seized by the technique. Retrospective Process Tracing could identify waste-objects that were not covered with a relatively high frequency by the waste literature, and was able to do so *while* saving time compared to traditional Retrospective Meetings. In addition, I have discussed to what extent we can classify this technique as a substitute to the meeting, or whether it should be considered a supplement, as well as the credibility, transferability, dependability and confirmability of the results.

What we are left with, is a procedure that might be adopted by other Agile development teams, which could serve as a tool for making Retrospective Meetings both more efficient and better suited as problem-solving arenas, rather than explanation-oriented settings. It is my belief that understanding waste, the concept and its implications, is important for any group of developers claiming to be *Agile* or *Lean*: Whether they adhere to the methods of DevOps, Kanban, Scrum, eXtreme Programming, or even a mix of all the above.

Finally, I'm happy with the study and its produced artefact. I hope that this thesis will fuel both a broader discussion on inefficiencies in software engineering, and more specifically on the concept of waste. As a concept, it has severe implications for the industry. I do hope that more literature will be produced to build bridges between practitioners and academics in the cross-section of organizational theory and software engineering, presenting practitioners with tools, grounded in research, to transform their processes. Hopefully, *Retrospective Process Tracing* will stand as an attempt to do just that.

References

Adams, C. (2016). What Is Google PageRank, How Is It Earned & Does It Matter in 2016? www.bruceclay.com/blog/what-is-pagerank/. Retrieved 26.05.17.

Ahimibisibwe, A., Cavana, R. Y., & Daellenbach, U. (2015). A contingency fit model of critical success factors for software development projects: A comparison of agile and traditional plan-based methodologies. Journal of Enterprise Information Management, 28(1), 7-33.

Almeida, R. (2014). Retrospective Wall Board: A Tool to Improve the Results of a Retrospective Meeting. www.scrumalliance.org/community/articles/2014/january/retrospective-wall-board. Retrieved 14.10.16

Ambler, S. W. (2005). Big Modeling Up Front (BMUF) Anti-Pattern. www.agile-modeling.com/essays/bmuf.htm. Retrieved 01.05.17.

Beach, D., & Pedersen, R. B. (2013). Process Tracing Methods: Foundations and Guidelines. Michigan, USA: The University of Michigan Press.

Beck, K., Beedle, M., Bennekum, A. v., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K, Sutherland, J. & Thomas, D. (2001). The Agile Manifesto. www.agilemanifesto.org. Retrieved 11.11.2015.

Bermejo, P. H. d. S., Zambalde, A. L., Tonelli, A. O., Souza, S. A., Zuppo, L. A., & Rosa, P. L. (2014). Agile Principles and Achievement of Success in Software Development: A Quantitative Study in Brazilian Organizations. Procedia Technology, 16, 718-727.

Blank, S. (2013). Why the Lean Start-Up Changes Everything. Harvard Business Review.

Brown, G. A. (2015). An Examination of Critical Success Factors of an Agile Project. (Doctor of Philosophy), Capella University, ProQuest.

Chow, T., & Cao, D.-B. (2008). A survey study of critical success factors in agile software projects. Journal of Systems and Software, 81(6), 961-971.

Collier, D. (2010). Process Tracing: Introduction and Exercises.

Darwish, N. R., & Rizk, N. M. (2015). Multi-Dimensional Success Factors of Agile Software Development Projects. International Journal of Computer Applications, 118(15), Multi-Dimensional Success Factors of Agile Software Development Projects.

Diebold, P., & Dahlem, M. (2014). Agile practices in practice: a mapping study. EASE '14 Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering.

Dingsøyr, T., Dybå, T., & Abrahamson, P. (2008). A Preliminary Roadmap for Empirical Research on Agile Software Development. Paper presented at the AGILE '08, Toronto, ON, Canada.

Evera, S. V. (1997). Guide to Methods for Students of Political Science. USA: Cornell University Press.

Fossgeim, H. J. (2009). Informert Samtykke. www.etikkom.no/FBIB/Temaer/Personvern-og-ansvar-for-den-enkelte/Samtykke/. Retrieved 03.02.17.

Gross, J. M., & McInnis, K. R. (2003). Kanban Made Simple: Demystifing and Applying Toyota's Legendary Manufactoring Process. NY, USA: AMACOM.

Guo, Y., Spínola, R. O., & Seaman, C. (2016). Exploring the costs of technical debt management - a case study. Empirical Software Engineering, 21(1), 159-182.

Gutierrez, K. (2014). Studies confirm the Power of Visuals in eLearning. info.shiftelearning.com/blog/bid/350326/Studies-Confirm-the-Power-of-Visuals-in-eLearning. Retrieved 24.05.17

Hartman, B. (2009). An Introduction to Planning Poker. dzone.com/articles/introduction-planning-poker. Retrieved 26.05.17.

Haugen, N. C. (2006). An Empirical Study of Using Planning Poker for User Story Estimation. Paper presented at the Agile Conference, 2006, Minneapolis, MN, USA.

Hedstrom, P., & Ylikoski, P. (2010). Causal Mechanisms in the Social Sciences. Annual Review of Sociology, 36, 49-67.

Hodgkinson, H. L. (1957). Action Research - A Critique. The Journal of Educational Sociology, 31(4), 137-153.

Hossain, E., Babar, M. A., & Paik, H.-y. (2009). Using Scrum in Global Software Development: A Systematic Literature Review Paper presented at the 2009 Fourth IEEE International Conference on Global Software Engineering.

Hummel, M., & Epp, A. (2015). Success Factors of Agile Information Systems Development: A Qualitative Study. Paper presented at the System Sciences (HICSS), 2015 48th Hawaii International Conference, Kauai, HI, USA

Ikonen, M., Kettunen, P., Oza, N., & Abrahamson, P. (2009). Exploring the Sources of Waste in Kanban Software Development Projects. Paper presented at the Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference, Lille, TBD, France.

Kelle, E. V., Wijst, P. v. d., & Wisser, J. (2015). An empirical study into social success factors for agile software development. Paper presented at the CHASE '15 Proceedings of the Eighth International Workshop on Cooperative and Human Aspects of Software Engineering

Kemmis, S., McTaggart, R., & Nixon, R. (2014). The Action Research Planner: Doing Critical Participatory Action Research.

Kropp, M., & Meier, A. (2015). Agile Success Factors - A qualitative study about what makes agile projects successful.

Matook, S., & Vidgen, R. (2014). Harmonizing critical success factors in agile ISD projects. Paper presented at the AMCIS 2014: 20th Americas Conference on Information Systems, Savannah, GA, USA.

Mills, G. E. (2000). Action Research: A Guide for the Teacher Researcher: Merill Prentice Hall.

Misra, S. C., Kumar, V., & Kumar, U. (2009). Identifying some important success factors in adopting agile software development practices. Journal of Systems and Software, 82(11), 1869-1890.

Monden, Y. (2012). Toyota Production System: An Integrated Approach to Just-In-Time. Boca Raton, FL, USA: Taylor & Francis Group, LLC.

Mueller, E. (2010, 07.12.2016). What is DevOps? www.theagileadmin.com/what-is-devops/. Retrieved 26.05.17.

Nikitina, N., Kajko-Mattson, M., & Stråle, M. (2012). From Scrum to Scrumban: A Case Study of a Process Transition. Paper presented at the ICSSP '12 Proceedings of the International Conference on Software and System Process Zurich, Switzerland.

Nord, R. L., Bellomo, S., & Ozkaya, I. (2013). A study of enabling factors for rapid fielding combined practices to balance speed and stability. Paper presented at the Software Engineering (ICSE), 2013 35th International Conference, San Francisco, CA, USA.

Oates, B. J. (2006). Researching Information Systems and Computing. London, UK: SAGE Publications Ltd.

Pedersen, M. (2013). A quantitative examination of critical success factors comparing Agile and waterfall project management methodologies. (Doctor of Philosophy), Capella University, ProQuest.

Peterson, D. (2009). Buffer your bottlenecks. www.kanbanblog.com/article/buffer-your-bottlenecks.html. Retrieved 26.05.17.

Poppendiek, M., & Poppendiek, T. (2003). Lean Software Development. Boston, USA: Addison-Wesley.

Reason, P., & Bradbury, H. (2001). Handbook of Action Research (1 ed.). London, UK: SAGE Publications.

Rohlfing, I. (2012). An Integrative Framework Process Tracing: Theory, Temporality, and Method: Palgrave Macmillan, UK.

Salsita Software (2015). The Shifting Definition of Front-End Developer. http://blog.salsitasoft.com/the-shifting-definition-of-front-end-developer/. Retrieved 26.05.17

Schwaber, K., & Sutherland, J. (2016). The Scrum Guide. www.scrumguides.org/scrum-guide.html. Retrieved 02.09.16.

Stankovic, D., Nikolic, V., Djordjevic, M., & Cao, D.-B. (2013). A survey study of critical success factors in Agile software projects in former Yugoslavia IT companies. Journal of Systems and Software, 86(6), 1663-1678.

Tanner, M., & Willingh, U. v. (2014). Factors Leading to the Success and Failure of Agile Projects Implemented in Traditionally Waterfall Environments. Paper presented at the Make Learn International Conference Porotoz, Slovenia.

Trochim, W. M. K. (2006, 20.10.2006). Qualitative Validity. www.socialresearchmethods.net/kb/qualval.php. Retrieved 15.05.17.

Tsirakidis, P., Kobler, F., & Krcmar, H. (2009). Identification of Success and Failure Factors of Two Agile Software Development Teams in an Open Source Organization. Paper presented at the Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference, Limerick, Ireland.

VersionOne. (2017). Measuring the Velocity of your Agile Scrum Team. www.versionone.com/agile-101/agile-management-practices/agile-scrum-velocity/. Retrieved 17.02.17.

Wan, J., & Wang, R. (2010). Empirical Research on Critical Success Factors of Agile Software Process Improvement. J. Software Engineering & Applications, 3, 1131-1140.

Wells, D. (1999). The Rules of Extreme Programming. www.extremeprogramming.org/rules.html. Retrieved 23.05.17.

Womack, J. P., & Jones, D. T. (1996). Lean Thinking. New York, NY, USA: FREE PRESS.

Wood, S., Michealides, G., & Thomson, C. (2013). Successful extreme programming: Fidelity to the methodology or good teamworking? Information and Software Technology, 55(4), 660-672.