European Master in System Dynamics

VULNERABILITY DYNAMICS

A Model-Based Case Study about the Interactions between Pressure in Agile Secure Software Development, Software Vulnerabilities, Adversarial Behaviour, and Attack Response: Trading Off Software Functionality and Software Security

Keywords: Vulnerability, Short-Term Business Risk, Long-Term Security Risk, Strategic Management, Theory Building, Cyber Security, Agile Secure Software Development, Pressure, Firefighting, Adversarial Behaviour, Complexity, Modelling, System Dynamics, Group Model Building

Author: Jonas Matheus
S4838297
Student of the European Master in System Dynamics

Document: Master Thesis
Masterthesis in System Dynamics (MTHEMSD)

1st Supervisor: Dr. Vincent de Gooyert
Assistant Professor Radboud University Nijmegen
2nd Supervisor: Dr. David Wheat
Professor University of Bergen

Date: 14 August 2017

ABSTRACT

To improve performance, organisations inside and outside the ICT sector buy, rent, borrow, and particularly develop own software solutions. At the same time, growing numbers of software vulnerabilities make software being the prime vector for malicious cyber attacks which disrupt business, cause disproportionate costs, and threaten the survival of organisations. Resources in software development are limited and organisations have to trade off between software functionality to cope with "time to market" pressure and software security to potentially fend off cyber attacks. Although it is known that trade-offs and subsequent stress cause defects which lead to vulnerabilities, no research has been conducted on the interaction between pressure, software vulnerabilities, external cyber attacks, and organisational attack mitigation. Hence, having been conducted as a model-based case study in a financial organisation in Europe, this research aimed to close this gap by investigating and explaining the influence of the interaction between pressure in software development, software vulnerabilities, external cyber attacks, and organisational attack response on the trade-off between software functionality and software security. In the end, this research led to the following seven contributions. First, the study shed light on the interaction between pressure, software vulnerabilities, cyber attacks, and attack mitigation. Second, by explicitly connecting pressure, defects, and vulnerabilities this study showed a potential pathway to successful cyber attacks. Third, this study explained the dilemma between fixing vulnerabilities fast to avoid successful exploitation and potential problems arising from firefighting due to fast problem solving. Fourth, the study described cyber adversaries as competitors which causes the need to integrate business, ICT, and cyber security strategies. Fifth, addressing both vulnerabilities and attacks leads to the potential of a dual firefighting mechanism with two apparent performance optima and one actual but lower one. Sixth, investigating the interactions described above enhanced understanding about the trade-off between software functionality and software security, and showed that initial short-term gains may be lost due to long-term insecurity. Finally, having generalised the outcomes of the research, this study provided testable propositions to take a first step in building an explicit theory of the dynamics of vulnerabilities, going beyond the case of secure software development and cyber security.

ACKNOWLEDGEMENTS

I would like to thank my four supervisors who have continuously supported, taught, challenged, and inspired me at different stages of the process. I would like to thank Dr. Guzay Pasaoglu for helping me to start this study and to aim high from the very beginning. I would like to thank Dr. Vincent de Gooyert for accepting the challenge to take over this work at a very late stage, for the tough inquiries that made me question my work, for the productive recommendations that helped me to get this study forward, and overall, for the enriching, fruitful, and enjoyable collaboration. I would like to thank Prof. Dr. Etiënne Rouwette for functioning as temporary supervisor, for introducing many valuable ideas, for paying attention to detail, and for helping me to round off this study. Finally, I would like to thank Prof. Dr. David Wheat for his flexibility and interest when being my second supervisor and examiner within the European Master in System Dynamics. In short, it has been a pleasure to work with all of you.

Moreover, I would like to thank all involved persons from the European Financial Organisation. I would like to thank the collaborating DevOps teams who provided me with extremely valuable insights. Next, I would like to especially thank the collaborating cyber security department and the responsible team. Thank you for supporting me in this project, for answering all my questions, and for treating me as your colleague. In short, it was a pleasure to work with you, to learn from you, to add value to this organisation, and to be your colleague.

Additionally, I would like to thank all of my fellows from the European Master in System Dynamics. I very much enjoyed working, learning, travelling, and living with and from you. It has been a pleasure to become part of this colourful and global family. Along the same line, I would like to thank all of my friends at home who have been waiting for me to come back for more than four years now. Thank you for being patient, for being there for me, and simply for being my friends over all these years. You have always been with me when I was abroad, no matter where I was.

Furthermore, I would like to thank my family without whom I would have neither started this endeavour, nor being able to carry it out. Thank you for inspiring me, for supporting me, for visiting me at all these places, for encouraging me to go my way, and for the never ending interest in what I do despite my difficulties in actually explaining it.

My final thank goes to my girlfriend. Thank you my love for having made this journey with me. You have been my anchor, my guide, my sparring partner, my challenger, my teacher, my beauty, my friend, my hope, and my home.

LIST OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Information and communication technology (ICT) increasingly represents the lifeline of almost all parts of the public and private sector (DNI, 2012; Leopold, Bleier, Skopik, 2015). Already since the 1980s, business organisations have recognised the opportunities of ICT and consequently built and strengthened their capabilities in planning, developing and operating it to sustain and enhance performance and competitive advantage to eventually achieve long-term success[1] (Amit, & Zott, 2001; Brynjolfsson, & Hitt, 2000; Henderson, & Venkatraman, 1993; Kettinger, Grover, Guha, & Segars, 1994; Porter, & Millar, 1985; Powell, & Dent-Micallef, 1997; Ravichandran, & Lertwongsatien, 2005; Wade, & Hulland, 2004). Since software enables endusers to actually employ ICT, next to renting or buying software applications from specialised vendors or using open source solutions, many organisations inside and lately also outside of the ICT sector develop and operate own software to improve their business (Wysopal, 2012). As one of the first, the financial sector has made technology a major priority (Johnston, & Carrico, 1988; Porter, & Millar, 1985) and financial organisations are on the way of becoming "technology companies with a banking licence" (FinExtra, 2017). While the specific benefits of ICT and software have been assessed very differently (e.g. Boehm, 1984; Kettinger et al., 1994; Powell, & Dent-Micallef, 1997), it is undisputed that ICT and "software [have become] ingrained in daily business activities" (Arora, Caulkins, & Telang, 2006, p. 465).

Next to the opportunities created by increasingly employing ICT and software, growing numbers of successful cyber attacks (Figure 1) affect the performance of organisations by evoking disproportionate costs (Anderson et al., 2013; Gillet, Hübner, & Plunus, 2010; Ponemon Institute, 2016; Telang, & Wattal, 2007) and even threaten their survival, as experienced by the Dutch firm DigiNotar which filed for bankruptcy as the consequence of a large scale cyber attack in 2011 (Arthur, 2011; Zetter, 2011). More generally, it was estimated that global costs due to malicious cyber activities ranged in 2013 between 300 billion to one trillion US-Dollars which is equal to 0,4 to 1,4 percent of the worldwide GDP (McAfee, 2013; Verizon, 2016).

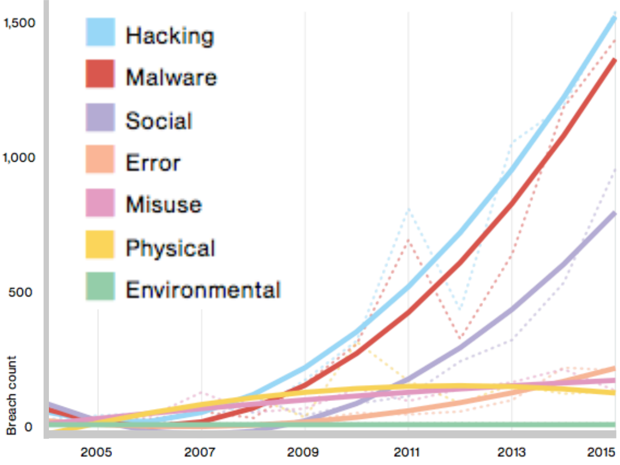


Figure 1: Number of Breaches per Threat Action Category (Verizon, 2016)

[1] Porter "assume[s] that firm success is manifested in attaining a competitive position or series of competitive positions that lead to superior and sustainable financial performance. Competitive position is measured [...] relative to the world's best rivals. [...] A successful firm may 'spend' some of the fruits of its competitive position on meeting social objectives or enjoying slack" (1991, p. 96).

For any kind of determined, strategically thinking, and malicious actor in the cyber space, software applications are a "prime vector into an organization" (Ahmad, 2007, p. 76) due to the growing number of known and unknown software vulnerabilities (Figure 2) as documented by several well-known sources, including the Verizon Data Breach Report (2016), the database on Common Vulnerabilities and Exposure (CVE) (CVE, 2017), the software vendor McAfee (2014), or authors from the RAND Corporation (Ablon, & Bogart, 2017). Software vulnerabilities describe weak points in a piece of software which are caused by defects in the underlying code or configuration. Such weaknesses are subject to potential exploitation by external adversaries[2] through malware (i.e., malicious software) or hacking attacks (i.e., the unauthorised modification of software),[3] potentially causing business disruption, data compromise, and financial and reputational losses (Anderson et al., 2013; Heitzenrater, Böhme, & Simpson, 2016; Landwehr, 2001; McGraw, 2006; Mohammed, Niazi, Alshayeb, & Mahmood, 2017; Pfleeger, Pfleeger, & Margulies, 2015).

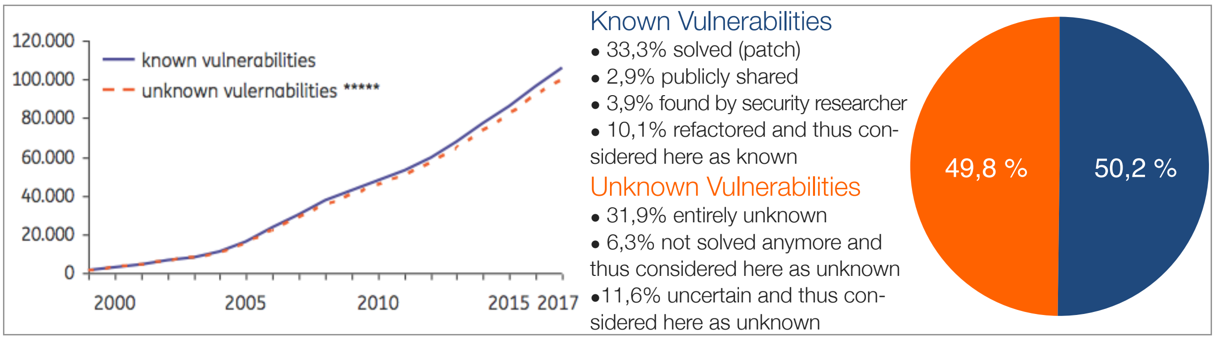


Figure 2: Cumulative Known and Unknown Vulnerabilities and Distribution. While the CVE Database only provides the amount of publicly known vulnerabilities, the researchers from the RAND Corporation found out the distribution of vulnerabilities. Combining the insights from the CVE and the RAND corporation leads to the conclusion that only 50 percent of the global software vulnerabilities are known to the public. Hence, there may be up to the double amount of total vulnerabilities worldwide (Developed by experts within the collaborating case study organisation and based on Ablon & Bogart, 2017, p. 28ff.; CVE, 2017).

Within the scope of common cyber security defence measures,[4] secure software development aims to avoid software vulnerabilities and consequently helps to prevent successful cyber attacks throughout the entire lifecycle of a software solution. A software lifecycle describes the five phases of initiation, development/acquisition, implementation/ assessment, operations/maintenance, and disposal (Kissel et al., 2008). Secure software development (also called secure software engineering) differs from functionality-oriented software in the sense that considerable extra effort is directed towards creating systems and applications that are as little vulnerable as possible through adjusted and more re-

[2] Next to external adversaries, particularly insider threats play an important role (e.g., Martinez-Moyano, Rich, Conrad, Andersen, & Stewart, 2008). However, due to their different nature and functioning insider threats are excluded here.

[3] Note, however, that the terms of hacking and hackers are not automatically meant in a pejorative sense. Instead, von Krogh, Rossi-Lamastra, and Haefliger (2012) connect the beginning of open source software with the hacker culture from the 1970s.

[4] Next to cyber security the terms information security, data security and computer security exist and have slightly different meanings (von Solms & van Niekerk, 2013). For the purpose of simplicity this research will always use the term cyber security except in citations or when another term is more applicable. For further information on typical cyber security measures see e.g. NIST, 2014.

source consuming practices. In this context, for instance, functionality aspects are tested regarding their intended use, whereas secure software engineering must focus on the almost infinite possibilities of misusing an application (e.g., Anderson, 2001; McGraw, 2006, 2012). Next to the obviously technological elements of secure software development, particularly behavioural and organisational aspects are important in explaining the phenomenon of software vulnerabilities because the reasons range from technical challenges and innovations, through lack of security awareness, poor practice in software development and managerial decision making, to purely economic reasons of accepting software vulnerabilities (Arora et al., 2006; Gordon, & Loeb, 2002; Heitzenrater et al., 2016; McGraw, 2006, 2012; Mohammed et al., 2017; Piessens, 2002; Rahmandad & Repenning, 2016; Shumba, Walden, Ludi, Taylor, & Wang 2006). In his seminal work on software engineering economics, Barry Boehm pointed out that "we deal with limited resources. There is never enough time or money to cover all the good features we would like to put into our software products" (1984, p. 4). Hence, although aiming for both, organisations are forced to trade off between short-term gains through functionality and long-term stability through security, both impacting performance and success (Becker, 2014; Heitzenrater et al., 2016; Neumann, 2012). While functionality creates certain immediate value by supporting business, the benefits from security "do not come from 'making something happen' by enabling a strategy or enhancing an operation, but from the prevention and/or reduction of potential losses caused by security breaches" (Huang, Hu, & Behara, 2008, p. 794). Hence, decision makers need to simultaneously address the short-term business risk arising from market pressure due to competition (e.g., Rahmandad, 2012) and the potential long-term security risk of attack pressure from malicious cyber adversaries (e.g., Becker, 2014; Neumann, 2012). Interestingly, literature has not provided clarity to reduce this tension. Instead, it provides evidence for both, limited investments in cyber security due to market pressure and constraints for defence on the one hand (Arora et al., 2006; Böhme, & Moore, 2009; Gordon, & Loeb, 2002), and comprehensive focus on security to fend off attacks, and also to improve overall performance, enhance software quality and lower costs on the other hand (Becker, 2014; Heitzenrater et al., 2016; McGraw, 2006; Neumann, 2012). While Neumann emphasises that "a well-reasoned understanding of the trade-offs is essential before potentially sacrificing possible future opportunities in an effort to satisfy short-term goals" (2012, p. 26), the business-driven perspective makes functionality of software solutions still constituting the heart of software development and deems security as an afterthought which can be sacrificed during in competition (Becker, 2014; McGraw, 2006, 2012; Neumann, 2012).

Such trade-offs between the short term (i.e., here functionality) and the long term (i.e., here security) have been widely discussed in the organisational theory and strategy literature (e.g. Laverty, 1996), and examples include inter alia exploitation and exploration (Levinthal, & March, 1993), or defect correction and process improvement (Repenning, & Sterman, 2002). Research has emphasised that balancing the short and long-term performance is crucial to the success and survival of an organisation (Levinthal, & March, 1993).

Although the tension between software functionality and software security blends in with the other examples of temporal trade-offs, to the best of the author's knowledge, there has been no investigation of this topic in the organisational theory and strategy literature.[5] Considering that trade-offs and subsequent pressure cause errors which lead to vulnerabilities (e.g., Austin, 2001; McGraw, 2006; Oliva, & Sterman, 2001; Rahmandad, 2005; Rahmandad, & Repenning, 2016; Repenning, & Sterman, 2002; Rudolph, & Repenning, 2002), it is surprising that previous research has addressed market pressure in software development (e.g., Arora et al., 2006), optimal investment in cyber security (e.g., Böhme, & Moore, 2009; Gordon, & Loeb, 2002), and the complexities of software engineering (see for a a very broad overview Cao, Ramesh, & Abdel-Hamid, 2010, p. 4), but not the interaction between pressure, software vulnerabilities, cyber attacks, and an organisation's response. Hence, this study aims to close this gap by investigating and explaining the dynamics of secure software development, software vulnerabilities, the malicious interference by cyber attacks of an external adversary, and organisational attack mitigation. Thus, this study addresses the following research question:

> *How does the interaction between pressure in software development, software vulnerabilities, external cyber attacks against an organisation, and the organisation's attempt to mitigate those attacks influence the trade-off between software functionality and software security?*

Starting with the phenomena of increasing software vulnerabilities and successful cyber attacks (Ablon & Bogart, 2017; CVE, 2017; von Kogh et al., 2012), this research is conducted as a model-based case study in a financial organisation in Europe (Perlow, & Repenning, 2009; Rahmandad, & Repenning, 2016). As usual in phenomenon-based and case study research, the phenomena and the broader literature are used to guide the data collection and analysis when seeking to answer the research question (von Kogh et al., 2012; Yin, 2014).

[5] For instance, a search in all fields for the terms "cyber security, "information security", and "secure software development" within high impact journals of organisational theory and strategy has revealed how little the topic is actually covered within this field. Academy of Management Journal shows zero, zero and zero results, Academy of Strategic Management Journal one, five and zero, Administrative Science Quarterly zero, one and three, Journal of Management zero, zero and zero, Management Science two, sixteen and zero, Organization Science zero, four and zero, and Strategic Management Journal zero, nine and zero.

Connecting the insights from integrating the different strands of literature across various fields[6] with the empirical findings, the study sheds light on the dynamic interaction between pressure, software vulnerabilities, cyber attacks, and organisational response which has the potential to causes several modes of persistent firefighting, wrong adaptation to the future, and escalatory patterns in adversarial behaviour. Considering this interplay enhances understanding about the trade-off between software functionality and software security. In so doing, the research presents practical implications for managers interested in secure software development and cyber security. Generalising the outcomes of the research (Yin, 2014), this study provides testable propositions to take a first step in building a theory of vulnerability dynamics, exceeding the case of secure software development and cyber security.

The remainder of this study is organised as follows: The research begins by giving an overview of relevant software development and security practices (2.1). Thereafter, it connects the different strands of literature to guide the case study research (2.2). Next, the study provides an overview of the methodology to make the process of theory building explicit (3.1), to describe and explain the data collection and analysis (3.2), and to present ethical considerations of this research (3.3). Afterwards, this study draws on the empirical findings from the research in the financial organisation to describe and explain the dynamic interactions in secure software development, software vulnerabilities, external cyber attacks, and organisational response (4). After answering the research question based on the findings (5.1), the study discusses practices for improving software development and security, thereby providing practical implications to managers in the field (5.2). Generalising the findings, this study presents testable propositions and necessary conditions to take a first step in building theory of vulnerability dynamics, unfolding beyond the fields of software engineering and cyber security (5.3). Based on this theory, theoretical implications are outlined (5.4). This study closes with summing up the insights and discussing rival theories, limitations and opportunities for future research (6).

2. THEORETICAL BACKGROUND

2.1 Software Development and Cyber Security

“Computer software continues to be the single most important technology on the world stage [… and has] become an indispensable technology for business, science,

[6] The different fields include organisational theory, organisational science, strategy, strategic management, system dynamics, management of information and communication technology, management of information systems, cyber security, information security, information risk management, security economics, (secure) software development, and (secure) software engineering.

and engineering” (Pressman, 2010, p. 2). Moreover, software enables the creation, change, and improvement of other technology (including software), and it is embedded in basically all forms of ICT. Research has shown that next to specialised software vendors, organisations from many other sectors, such as the financial industry, develop their own software solutions, resulting in more than 70 percent of internally developed software which was not purchased or rented from a software vendor (Wysopal, 2012). While, software has left the niche of the ICT sector, and has become elemental for any kind of organisation, it also has become a crucial vulnerability of an organisation’s ICT (Ahmad, 2007). This subsection first outlines software development and then turns towards relevant security practices.

2.1.1 Software Development

Software evolved more than sixty years ago and has gone through many changes, including its way of development. Broadly speaking, software engineering describes the activity of planning, developing, operating, and maintaining software through its entire lifecycle, and, since the late 1960s, is guided by so called software process models (MacCormack, Kemerer, Cusumano, & Crandall, 2003; Pressman, 2010). “Such process models are one of the most fundamental aspects of software development, governing the inclusion, frequency, timing and scope of development activities” (Heitzenrater et al., 2016, p. 2). While the framing of the various activities differs, they generally include the steps of requirements analysis, planning, design, development (i.e., writing the code), testing, deployment, operation, and decommission (Boehm, 1988; Heitzenrater et al., 2016; MacCormack et al., 2003; Pressman, 2010). Standard process models are spread out over a large continuum of different methods and range from rather plan-driven, sequential approaches, such as waterfall (Figure 3), at the one extreme, to flexible lightweight methods, such as agile development (Figure 4), at the other extreme (Boehm, 1988; Boehm, & Turner, 2005; MacCormack et al., 2003; Pressman, 2010). The waterfall model emphasises objectives, planning, control, and discipline, making it a rigorous and sound, but rather slow process. While widely believed that the probability of defects in software is lower when following such plan- and control-driven methods, MacCormack et al. (2003) provide evidence that more flexible approaches compensate for problems of rigour by obtaining

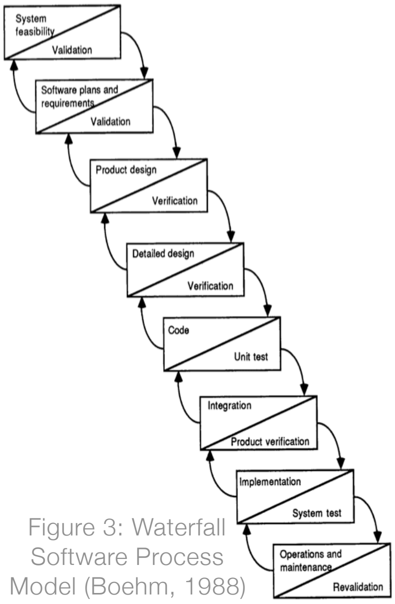


Figure 3: Waterfall Software Process Model (Boehm, 1988)

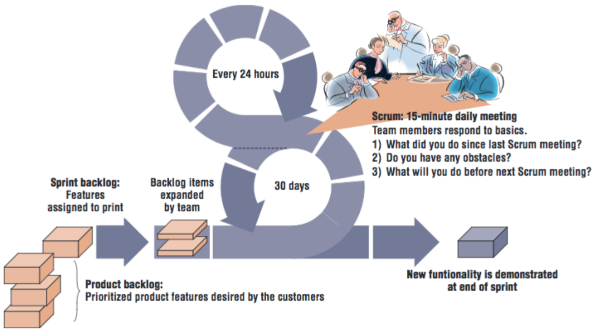

Every 24 hours
Scrum: 15-minute daily meeting
Team members respond to basics.
1) What did you do since last Scrum meeting?
2) Do you have any obstacles?
3) What will you do before next Scrum meeting?
30 days
Sprint backlog: Features assigned to print
Backlog items expanded by team
Product backlog: Prioritized product features desired by the customers
New funtionality is demonstrated at end of sprint

Figure 4: Agile Software Development Process (Boehm, & Turner, 2005, p. 33)

fast customer feedback.[7] In the same vein, the slowness of the waterfall model makes it less able to account for the "software industry's increasing needs for rapid development and [for coping] with continuous change" (Boehm, & Turner, 2005, p. 30) than flexible approaches, such as agile development (Boehm, & Turner, 2005; Cao et al., 2016). "In general, agile methods are lightweight processes that employ short iterative cycles, actively involve users to establish, prioritize, and verify requirements, and rely on team's tacit knowledge as opposed to [the lengthy] documentation" (Boehm, & Turner, 2005, p. 32) of other methods such as waterfall. As such, agile teams are self-organised which allows them to adjust their work to current demand and resource availabilities, generally aiming to match the customers expectations and avoid high work pressure. Agile teams employ fast release cycles of less than a month, also known as a sprint, in which they deliver small but full pieces of software with a complete subset of functionalities (Boehm, & Turner, 2005; MacCormack et al., 2003; Pressman, 2010). Next to creating immediate business value, the fast feedback cycles also enable to, firstly, decrease time to market, and thereby improve competitive advantage (Arora et al., 2006), and secondly, discover mismatching customer demands, and thereby reduce costs which, as shown by Boehm (1984) or Stecklein and colleagues (2004), escalate exponentially throughout the software lifecycle.[8] To this end, it is the ability to react on rapidly changing environments and customer demands, and to create immediate business value due to fast releases that caused agile approaches to succeed over sequential methods in software development.[9]

2.1.2 Secure Software Development

"Software security is the idea of engineering software so that it continues to function correctly under malicious attack" (McGraw, 2012, p. 662). Cyber adversaries commonly conduct their attacks by attempting to exploit software vulnerabilities through malware and hacking attacks, such as in the recent cases of WannaCry and NotPetya that affected

[7] Note that Cormack et al. (2003) emphasise that any process model is only successful if it is applied consistently and not in a "cherry-picking-piecemeal" fashion. They emphasise that "to the degree that such a process relies on a coherent system of practices, a piecemeal approach is likely to lead to disappointment" (2003, p. 84).

[8] The costs of changes after the requirements phase increase according to Stecklein and colleagues (2004) as follows: Design = 5x - 7x, Develop = 10x - 26x, Test = 50x - 177x, and Operations = 100x - 1000x.

[9] To the interested reader, particularly Pressman's (2010) work covering many aspects of software development is recommended.

hundreds of thousands of computers worldwide (Fox-Brewster, 2017). Next to technical innovation which inevitably opens new paths of exploitation (Ahmad, 2007), "humans play a central role in security measures" (Proctor, & Chen, 2015, p. 721) and are often considered as the weakest link (i.e., the least protected point) in cyber security (Bulgurcu, Cavusoglu, & Benbasat, 2010; Lineberry, 2007). In this context, some of the biggest problems in software security are the lacking security awareness of developers and operators (DevOps), limited development skills and knowledge in the field of security, or missing compliance to cyber security rules (McGraw, 2012). To address human weaknesses in software engineering, secure software development relies on training and applying a broad range of technical tools (e.g., Metasploit), specific practice recommendations (e.g., OWASP Top 10), and security process models which are combined with the process models described above (Ahmad, 2007; Heitzenrater et al., 2016; de Win, Scandariato, Buyens, Grégoire, & Joosen, 2008).[10] To this end, all of the security process models underline that "security is not a feature that can be added to software […]. Security is an emergent property of a system" (McGraw, 2006, p. 213) that evolves throughout the entire lifecycle (Figure 5). It is commonplace that reducing the introduction of vulnerabilities prior to the release of software has several major benefits: First, it is a major step in improving overall cyber security as many forms of attacks rely on exploiting this type of weaknesses. Second, software security is part of general software quality. In contrast to general quality assurance though, security testing demands to think and act like a malicious attacker. Hence, increasing quality may help to improve security, but enhancing security always results in higher quality. Finally, building security in is much more cost effective than any security measure taken after deployment (Heitzenrater et al., 2016; McGraw, 2006, 2012).[11]

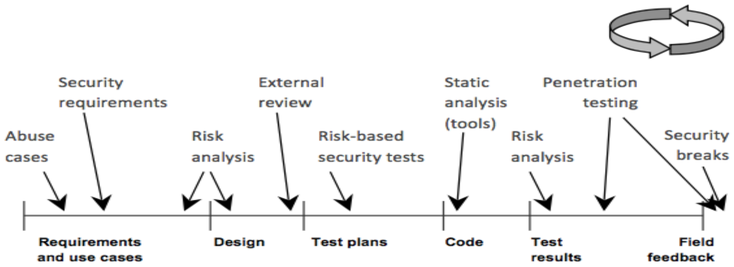


Figure 5: Software Security Best Practice applied throughout the Lifecycle. Although the stages appear to be sequential like in the Waterfall-Model, generally organisations follow an iterative approach, such as agile development, and thus, apply these practices over and over again. (McGraw, 2012, p. 663)

2.2 Pressure arising from Trade-Offs between Functionality and Security

Despite these documented benefits of accounting for security from early on and throughout the entire software development lifecycle, the overall security level of ICT

[10] Common security practices and process models include Adobe SPLC (2016), Microsoft SDL (2017a, 2017b), OWASP Top10 and OWASP Clasp/SAMM (2013, 2016, 2017), or McGraw's SSDL Touchpoints (2006, 2012; also McGraw, Migues, & West, 2016).

[11] To the interested reader, particularly McGraw's (2006) work concerning secure software development is recommended.

and software has never been considerably increased, and instead, the number of software vulnerabilities, and subsequently successful cyber attacks, has been continuously growing (Bojanc, & Jerman-Blazič, 2008; McGraw, 2006, 2012; Verizon, 2016). Accordingly, "the overall question arises, why software vendors do not make their products more secure [in] the first place. The answer lies in economics" (Bojanc, & Jerman-Blazič, 2008, p. 415).

2.2.1 Capabilities in Information and Communication Technology

Generally speaking, organisations employ ICT and buy, internally develop and operate software to enhance performance and achieve competitive advantage. While earlier studies indicated a direct and positive link between ICT and performance, later research described contingent effects of technology (Wade & Hulland, 2004). In this sense, several authors described ICT as a strategic necessity to avoid suffering from competitive disadvantage compared to other organisations (Powell, & Dent-Micallef, 1997; Ravichandran, & Lertwongsatien, 2005). Similarly, studies emphasised that not merely possessing but fully integrating an organisation's core activities with ICT improves performance, creates business value, causes competitive advantage, and finally enables sustained success (Bharadwaj, 2000; Brynjolfsson, & Hitt, 2000; Henderson, & Venkatraman, 1993; Kettinger et al., 1994). Throughout the strategy literature, success was initially associated with strategies addressing an organisation's external environment (e.g., Porter, 1991). However, unstable and rapidly changing external environments caused a shift towards an organisation's internal resources and capabilities. According to Winter (2003), capabilities describe learned and continuously practiced activities that allow an organisation to improve the pursuit of their core tasks and objectives to achieve competitive advantage and success. In this context, literature has distinguished between operational capabilities which are "those that permit a firm to 'make a living' in the short term" and dynamic capabilities that "operate to extend, modify or create ordinary capabilities" (Winter, 2003, p. 991) in order to achieve success in the long-term (Collis, 1994; Eisenhardt, & Martin, 2000; Rahmandad, 2012; Teece, Pisano, & Shuen, 1997). While the distinction between operational and dynamic capabilities depends on the specificity of the issue and the core task of an organisation,[12] for companies outside of the ICT sector, planning, developing, integrating, securing and operating ICT rather describes a dynamic capability because it is done to extend, modify and create the way of how they make a living (Brynjolfsson, & Hitt, 2000; Henderson, & Venkatraman; Wade, & Hulland, 2004).

[12] Operational and dynamic capabilities are locally defined (Winter, 2003). Product development (including software development) is a dynamic capability (Rahmandad, 2012), but for a firm that develops software to make a living, this could be an operational capability.

In recent years, particularly financial organisations, such as Citigroup or the Norwegian bank DNB, have invested in building and strengthening their ICT related capabilities, including software development and operations, to fend off the attacks from technology start-ups that offer financial services (Dapp, 2014; FinExtra, 2017; Gandel, 2016). This development of "matching a firm's resources and capabilities to the opportunities that arise in the external environment" (Grant, 2010, p. 122) is the core of strategy and has been particularly dominant in the interaction of ICT with environments that are governed by rapid change and competitive pressure, such as the airline industry and the financial sector (Johnston, & Corrico, 1988; Porter, & Millar, 1985; Rivard, Raymond, & Verreault, 2006). Since ICT related capabilities take significant time to change and provide benefits with very different time delays (Brynjolfsson, & Hitt, 2000; Rahmandad, & Repenning, 2016; Ravichandran, & Lertwongsatien, 2005), "allocating a limited investment flow among them leads to inter temporal trade-offs, which are at the heart of executives' challenges" (Rahmandad, 2012, p. 138).

2.2.2 Time to Market, Software Economics and Security

Financial organisations develop software to modify and extent the core task of providing financial services and to improve their overall business activities. Taking a more nuanced view, capabilities in software development may be distinguished between creating software functionality and ensuring software security. These two development capabilities have very different temporal and financial implications. On the one hand, allocating resources to develop functionality of software leads to immediate benefits to customers and creates value for the organisation. Hence, developing software functionality within short sprints permits a financial organisation to directly address market pressure, and thus, pays off with very short time delays (Arora et al., 2006; Boehm, & Turner, 2005; Pressman, 2010). On the other hand, the effects of software security are much more uncertain. Focusing on software security implies considerable additional development effort to potentially prevent unknown future cyber attacks (Huang et al., 2008). Next, the absence of known attacks does not automatically mean that an organisation is secure and no attacks have occurred, but potentially also that attacks have not been detected and yet taken place. Mistakenly perceiving a low cyber security risk because of few detected attacks, organisations think themselves safe and decrease future cyber security investments, thereby reinforcing the erroneous sense of security (Martinez-Moyano, Conrad, & Andersen, 2011). Finally, even if organisations know that security measures have prevented and/or reduced potential losses, it is often unknown which security measure has proven to be effective, meaning that the overall value of security measures is difficult to quantify (for some approaches to the economics of cyber security see e.g., Anderson et al. 2013, Gordon, & Loeb, 2002; Heitzenrater et al., 2016). In the end, decision makers need to simultaneously address the short-term business risk of market pressure from competitors through enhancing software functionality (Arora et al., 2008) and the potential long-term security risk of attack pressure from malicious cyber adversaries through software security (Becker, 2014; McGraw, 2006, 2012; Neumann, 2012). Too much focus on security impairs performance and success, whereas too little focus on security may cause software vulnerabilities and subsequent successful cyber attacks (Broderick, 2001; McGraw, 2006). Despite the acknowledged need for a balance between software functionality and software security, companies rather sell their software first and fix it later. In this sense, it is common to trade-off the long-term quality, robustness, and security of software against the short-term gain from releasing functionalities (Arora et al. 2008; Becker, 2014; Neumann, 2012).

2.2.3 Temporal Trade-Offs in Strategy

The topic of temporal trade-offs between the short term and the long term received particular attention in the organisational theory and strategy literature (e.g. Laverty, 1996). Next to investments in operational and dynamic capabilities (Rahmandad, 2012; Rahmandad, Henderson, Repenning, 2016; Winter, 2003), examples included the previously mentioned exploitation and exploration (Levinthal, & March, 1993; Walrave, van Oorschot, & Romme., 2011), as well as defect correction and process improvement (Repenning, & Sterman, 2002). Additionally, studies covered the topics of direct and supporting activities (Porter, 1991), production and protection (Goh, Love, Brown, & Spickett, 2012), reactive and preventive maintenance (Sterman, 2000), or performance and robustness (Rahmandad, & Repenning, 2016). In practice, strategy and decision making have appeared to favour "short-termism" (Laverty, 1996, p. 825) which may be explained by an organisation's struggle for survival (Rahmandad, 2012), a favourable balance between operational and dynamic capabilities that allow reaping the rewards (Rahmandad et al., 2016), stock market pressure and discounting of the future (Laverty, 1996), managerial myopia (i.e., "the tendency to overlook distant times, distant places, and failures" (Levinthal, & March, 1993, p. 95)), humans' difficulty in understanding dynamic complex systems and disruptive events (Rudolph, & Repenning, 2002; Sterman, 1994, 2000, 2006), mutual attribution errors in an environment governed by time delays (Repenning, & Sterman, 2002), or the fast search for an optimal allocation of fungible resources in a slowly adjusting system (Rahmandad, & Repenning, 2016). In the end, both, the short term and the long term, are equally important as “an organization cannot survive in the long run unless it survives in each of the short runs along the way, and strategies that permit short-run survival tend to increase long-run vulnerability” (Levinthal, & March, 1993, p. 110).

Since resources in software development are limited (Boehm, 1984; Ethiraj, Kale, Krishnan, & Singh, 2004), trading off short-term benefits from functionality to address market pressure and long-term robustness from security to cope with cyber attacks results in pressure resting on DevOps and software engineers. As described by Austin (2001) and Rahmandad and Repenning (2016), pressure is a major reason for errors in software development. Since errors may turn into vulnerabilities once the software is released, pressure should be a major security concern. Interestingly though, research has not investigated the connection between pressure, software defects, software vulnerabilities, and cyber attacks.

2.2.4 Pressure in Software Development and Software Vulnerabilities

Several recent studies indicated the mixed impact of pressure on performance and errors in production and service (see for example Goh et al., 2012; Oliva, & Sterman, 2001; Perlow, Okhuysen, & Repenning, 2002; Rahmandad, & Repenning, 2016; Repenning, & Sterman, 2002; Rudolph, Morrison, & Carroll, 2009; Rudolph, & Repenning, 2002). Of particular interest in this context has been the Yerkes-Dodson Law (1908) which describes an inverted u-shaped relationship between pressure and performance (Figure 6). While having been controversial for a long time due to its potential lack of applicability in other contexts than electroshocked mice (see for a short discussion and applicable contexts for instance, Rudolph, & Repenning, 2002, p. 9), recent research provided strong evidence, supporting the claims of the inverted u-shaped relationship between pressure and performance (Lupien, Maheu, Tu, Fiocco, & Schramek, 2007).

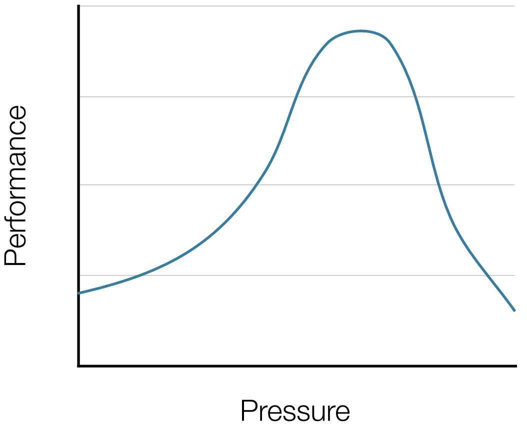


Figure 6: Example Shape of the Yerkes-Dodson Law (created by the Researcher, based on Rudolph, & Repenning, 2002; Rahmandad, & Repenning, 2016).

Particularly studies investigating the relationship between pressure and performance regarding the dynamic complexity within a production or service system relied on the Yerkes-Dodson Law (see e.g., Rudolph, & Repenning, 2002; Rahmandad, & Repenning, 2016; or Sterman, 2000). Sterman (2000, 2006) described dynamic complexity as the frequently counterintuitive behaviour of complex systems that arises from the interaction of its elements over time. Being in a constant state of change, any kind of dynamic complex system evolves unpredictably and adapts to new situations, no matter whether those are desirable or not. Most importantly, the effects of actions taken in such a system are generally subject to systemic (nonlinear and delayed) feedback (Forrester, 1971; Meadows, 2009; Sterman, 2000, 2002, 2006). Simply put, feedback is "a process in which action and information in turn affect each other" (Vennix, 1996, p. 31). In light of these system characteristics, understanding dynamic complexity constitutes a major challenge for humans and learning in such an environment is hampered by several barriers (Sterman, 1994). Combined with biases and heuristics, discrepancies in mental theories, and humans' bounded rationality decision making in dynamic complex systems is error-prone (Braun, 2002; Eisenhardt, & Zbaracki, 1992; Simon, 1985; Sterman, 2000, 2006; Tversky, & Kahneman, 1974, 1986; Vennix, 1996).

While not explicitly relying on the Yerkes-Dodson Law, Burchill and Fine (1997) investigated the effects of pressure on the quality of and errors within a development project. The authors found that market-oriented development leads to high quality products and little rework, whereas following "time to market" pressure results in a vicious circle of creating more pressure despite attempting to resolve it. Along the same line, Repenning and Sterman's theory of capability traps described challenges among the implementation of process improvement programmes which are "rooted in the ongoing interactions among the physical, economic, social, and psychological structures" (2002, p. 292) of the internal and external environment of an organisation. Similar to Burchill and Fine, managers' attempts to resolve pressure by increasing throughput eventually exacerbates the situation due to capability erosion caused by a lack of process improvement activities. Likewise, Rahmandad and Repenning (2016) investigated capability erosion arising from demand pressure and mistaken attempts of adapting to future workload, aiming for a fast and optimal allocation of fungible resources in a slowly adjusting system. Being based on the Yerkes-Dodson Law, this study advanced the concept by adding a real and a believed pressure-performance relationship, making it even more likely for an organisation to collapse. Slightly different, Goh and colleagues (2012) investigated organisational accidents caused by decreasing risk perception and increasing production pressure. In the end, all of the four studies recommended to decrease pressure by stepping back from the situation to learn about it and accepting short-term difficulties in order to achieve long-term success. In contrast, the results of Rudolph and Repenning's (2002) study on disasters arising from external pressure caused by interruptions exogenous to an organisation provided evidence that there are situations in which learning actually exacerbates the undesired development. The study, also based on the Yerkes-Dodson Law, showed instead that immediate response is necessary in order to prevent organisational collapse.

In summary, all of the studies described the endogenous connection between pressure and performance problems. The studies differ in the sense that some illustrated pressure through the application of the Yerkes-Dodson Law (Rahmandad, & Repenning, 2016; Rudolph, & Repenning, 2002), others took pressure for granted and rather focused on the misperception of feedback when taking decisions (Goh et al., 2012; Repenning, & Sterman, 2002), and others treated the decision of choosing pressure explicitly (Burchill, & Fine, 1997). Additionally, the studies offered different solutions to addressing pressure: Most explained the exacerbating effect of attempted problem solving in the short-term, such as increasing workload, whereas one study explicitly pointed out the need to immediately solve the issue in order to survive the situation (Rudolph, & Repenning, 2002). Interestingly, while all of the described studies investigated the endogenous creation or facilitation of organisational collapse, none of the studies included the escalating relationship between an organisation and a malicious actor from the organisation's external environment who aims to exploit the problems created within the organisation. Considering research on adversarial dynamics in the field of terrorism and security (Martinez-Moyano, Oliva, Morrison, & Sallach, 2015), escalatory patterns of behaviour are, however, common in the interaction between defenders and attackers. To the best of the author's knowledge, there have been no studies which connect organisational collapse caused by the relationship between pressure and performance with the interference of an external malicious actor. Hence, this study builds on the previously mentioned research, and investigates the exploitation of endogenously created performance issues and weaknesses within an organisation by a malicious external adversary which eventually results in an escalatory attacker-defender-interaction.

3. METHODOLOGY AND DATA

3.1 Model-Based Case Study for Theory Building in Complex Environments

This study considered the tension between software functionality and software security by investigating the interaction between work pressure, software vulnerabilities, cyber attacks, and organisational response to afterwards generalise its findings for making a first step in building an explicit theory of vulnerability dynamics. According to Kopainsky and Luna-Reyes, "theory can be understood as a coherent description, explanation and representation of observed or experienced phenomena […] and theory building, in turn, is the ongoing process of producing, confirming, applying, and adapting theory" (2008, p. 472f.). Generally speaking, literature suggested several ways to contribute to theory, such as grounding theory in data, building theory from theory, testing previously developed theoretical concepts, or expanding the extant theory by combining building and testing (Colquitt, & Zapata-Phelan, 2007; Davis, Eisenhardt, Bingham, 2007; Strauss, & Corbin, 1994; Vaughan, 1992; Yin, 2014). More specifically, as pointed out by Rudolph and colleagues (2009), it is common in system dynamics to rely on all of the previous possibilities and to build, test and advance theoretical concepts based on empirical insights, previously developed theory, or a combination of both (see for example Black, Carlile, & Repenning, 2004; Burchill, & Fine, 1997; Goh et al., 2012; Oliva, & Sterman, 2001; Perlow et al., 2002; Perlow, & Repenning, 2009; Rahmandad, 2012; Rahmandad, & Repenning, 2016; Repenning, & Sterman, 2002; Rudolph et al., 2009; Rudolph, & Repenning, 2002; Sastry, 1997). System dynamics is a scientific approach for understanding, analysing, modelling and simulating dynamic complex physical and social systems to deliver policy options, support decision making, or contribute to theory (e.g., Forrester, 1958, 1961; Kopainsky, & Luna-Reyes, 2008; Sterman, 2000). While the larger part of theory-oriented studies in system dynamics were based on quantitative approaches, a number of qualitative studies built theory by combining system dynamics with grounded theory or case study research (Azoulay, Repenning, & Zuckerman, 2010; Burchill, & Fine, 1997; Goh et al., 2012; Martinez-Moyano, McCaffrey, & Oliva, 2014; van Oorschot, Akkermans, Sengupta, & Wassenhove, 2013; Perlow, Okhuysen, & Repenning, 2002; and Repenning, & Sterman, 2002). Grounded theory and case study research are particularly useful in supporting system dynamics because they provide rigorous ways to identify emerging patterns, describe causal relationships and explain complex phenomena (Forrester, 1992; Kopainsky, & Luna-Reyes, 2008; Yin, 2014). Case studies provide the additional benefit of "increasing the generic nature of a system dynamics model" (Kopainsky, & Luna-Reyes, 2008, p. 478) through theoretical/analytical generalisation (i.e., building theory by continuously and iteratively comparing the emerging generic structure about the phenomenon to be explained with literature or data in a process of (dis-) confirmation).

Hence, this study takes the phenomenon of growing numbers of software vulnerabilities and cyber attacks as a starting point to investigate vulnerability dynamics. Following the examples of qualitative theory building in system dynamics, this research integrates system dynamics, case study research and phenomenon-based research. The study describes an iterative process of continuously comparing empirical insights and knowledge from literature which fosters the process of generalising findings and thereby building a dynamic theory to explain the observed phenomena (Figure 7) (Burchill, & Fine, 1997; von Kogh et al., 2012; Kopainsky, & Luna-Reyes, 2008; Sutton, & Staw, 1995; Yin, 2014). Considering the dynamic complexity of an organisation's software engineering process and its interaction with external adversaries, integrating system dynamics, case study research and phenomenon-based research is particularly useful. Firstly, there is a growing appreciation in case study research for studying complex issues (Anderson, Crabtree, Steele, & McDaniel Jr. 2005), and secondly, all of the three methods are powerful in addressing multifaceted, interrelated, and dynamic complex phenomena (von Kogh et al., 2012; Kopainsky, & Luna-Reyes, 2008; Sterman, 2000; Yin, 2014).

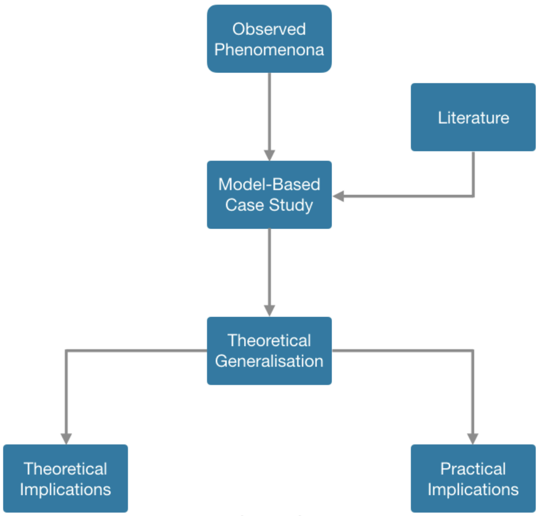


Figure 7: Model-Based Case Study Research Process for Theory Building with System Dynamics (based on Luna-Reyes & Andersen, 2003 typical steps in system dynamics research; Kopainsky, & Luna-Reyes, 2008 integrating system dynamics and case studies; and Yin, 2014 practices in case study research).

3.2 Case Selection

Having conducted the case study in a financial organisation in Europe had several benefits for the investigation at hand: First, financial organisations are subject to particularly high cyber risk (De Nederlandsche Bank, 2015, 2016; Deloitte, 2016; National Cyber Security Centre, 2016). Second, they have to bear the highest costs of cyber attacks throughout all industries (Ponemon Institute, 2016). Third, financial organisations increasingly rely on ICT due to financial gains and develop their own software solutions (Bauer, & van Eeten, 2011; Johnston, & Carrico; Porter, & Millar, 1985). Finally, they are amongst others considered as part of critical infrastructure (Cabinet Office, 2010a, 2010b).[13] Hence, having studied their case in secure software engineering and the interferences from external cyber attacks provided particularly valuable insights for understanding the interaction between work pressure, software vulnerabilities, cyber attacks, and organisational attack mitigation.

Next to financial organisations being generally appropriate for the investigation at hand, having conducted the case study in the collaborating European financial organisation was particularly suitable: Due to the rapid business environment of the financial sector, the organisation generally develops software following an agile approach, and also other, non-technical teams conduct their work according to the agile methodology

[13] The UK Cabinet Office defined critical infrastructure as "those infrastructure assets (physical or electronic) that are vital to the continued delivery and integrity of the essential services upon which [a country] relies, the loss or compromise of which would lead to severe economic or social consequences or to life loss" (2010a, p. 8). For information on cyber security in critical infrastructure see e.g. Miller & Rowe, 2012.

in order to flexibly address the internal and external environment. The organisation uses external software from third parties, such as commercial off-the-shelf software (bought), software as a service (rented), and open source software (borrowed), and has a strong focus on internal software development and operations. Within the organisation, mainly DevOps take care of third party and internally developed software throughout the entire lifecycle. They collaborate on this task with more specialised software engineers, system architects, the security community, and internal and external customers. Finally, they are part of development, operations, and emergency response activities in case of an attack. This ability to flexibly switch tasks is particularly interesting in cases of pressure as pointed out by Rahmandad and Repenning (2016). Since agile approaches were implemented within the organisation several years ago, most of the teams are rather mature in software engineering, and there is a growing commitment to address security concerns.

Throughout the case study, the author spent on average three days a week on site over the course of six months. The outcomes of the study will be used within the financial organisation for operational use and strategic decision making. Consequently, the researcher was considered as a team member within the organisation and received full support for his work during the period of the collaboration.[14]

3.3 Data Collection and Analysis

Data within the financial organisation was mainly collected through the application of group model building, a participatory approach of system dynamics involving stakeholders into the modelling process for improving problem structuring, knowledge elicitation, consensus building, analysis, and decision support (Vennix, 1996). Conducting group model building workshops as a data gathering method similar to focus groups (e.g., Gill, Stewart, Treasure, & Chadwick, 2008; Kopainsky, & Luna-Reyes, 2008; Morecroft, 1992; Morecroft, Lane, & Viita, 1991) has clear advantages over traditional interviews and was thus employed in the project. In contrast to individual interviews, the qualitative case data gathered from group model building is richer and more accurate because it is discussed systematically between the participants. Inaccuracies which are the uninvited companion of any abstraction, are more likely to be discovered throughout the workshops because of the precise nature of a system dynamics model. Creating a model serves as a group memory and means translating the mental database of the participants into the model for discussing and analysing it from a

[14] While the case study organisation covered the travelling expenses of the researcher, partly organised meetings and data, and provided a laptop with necessary software and further material, the author was not paid by the organisation. The contribution of the researcher to the organisation goes beyond this study but is not displayed here due to necessary confidentiality.

systemic perspective. In the end, mutually agreeing on the specific variables and links in the model leaves little room for later misinterpretation or wrong analysis of the qualitative case data and serves thereby as a first step for increasing the case study's internal validity (Forrester, 1992; Scott, Cavana, & Cameron, 2015; Vennix, 1996; Vennix, Andersen, Richardson, Rohrbaugh, 1992; Zagonel, 2002). Interestingly, group model building represents an approach which combines data collection and data analysis. First, throughout the workshops the knowledge from participants is collected and translated into a causal diagram. According to Merriam (2009), linking ideas, concepts, or categories in a meaningful way, for instance in such a model, represents the highest and most abstract level of data analysis. While also the links in a model obviously require further investigation, group model building yet describes a unique approach of data collection and analysis.

Over the course of one month, three participatory system dynamics workshops of three hours each took place on site and initially involved seven and in the second and third session five experts from different departments within the financial organisation.[15] The participants were chosen on the basis of their knowledge about the organisation's secure software development and cyber security system and were invited by the collaborating cyber security department and the researcher. As common practice, the workshops included a wide range of activities,[16] the overall topic was split into several smaller pieces (submodels),[17] the workshops were based on scripts commonly employed in group model building (Andersen, & Richardson, 1997; Luna-Reyes et al., 2006),[18] and the actual modelling exercises were always started with a preliminary model created by the researcher (Vennix, 1996). These preliminary models were based on the insights from literature and preparatory discussions with the gatekeeper and relied as much as possible on structures typical in system dynamics to increase the model's robustness, accuracy and ease of interpretation.[19] The models created with the participants during the workshops

[15] The participants covered the areas of ethical hacking, fraud, penetration testing, responsible disclosure, software development, system architecture, and vulnerability scanning. The number of participants changed because not all participants could take part in all sessions. The gatekeeper and a colleague of the author, both experienced in system dynamics and group model building, supported the researcher in the sessions. The colleague functioned as assistant and recorder within the sessions (see Appendix II. A)

[16] Overall, the workshops included the following activities: presenting the problem, explaining the methodology, addressing the topic by building the submodels, reviewing the previous sessions, discussing possibilities for measuring improvements, and at the end, reviewing the entire model created in the workshops in order to check and examine the connections between the different submodels, and discussing potential policy options.

[17] The submodels covered the topics of software development, third party software, DevOps, training and awareness, vulnerabilities, responsible disclosure, and adversary behaviour and attacks.

[18] The scripts employed throughout the three workshops were partly used with or without adjustments and include the following: Scheduling the day; logistics and room set up; creating a shared vision of a modelling project (only description elements used); nominal group technique; variable elicitation; causal mapping with seed structure; concept model; ratio exercise; model review; next steps and closing; initiating and elaborating a causal loop diagram; reflector feedback.

[19] Studies included Rahmandad & Repenning (2016) about software development, errors and wrong managerial adaptation; Oliva & Sterman (2001) and Rudolph & Repenning, (2002) about overtime, fatigue, corner cutting, and errors; Gonçalves, Hines, & Sterman (2005) about lean manufacturing; Repenning, & Sterman, (2002) about process improvement; Martinez-Moyano et al., (2015) about adversarial dynamics in terrorism, Rahmandad, & Hu (2010) about different formulations of the rework cycle; and Sterman (2000) for further standard approaches such as diffusion models or ageing chains and co-flows.

were cleaned and translated to the computer by the researcher immediately after the sessions. Decisions about how to understand, improve and explain the model were guided by the analytic technique of explanation building common in case study research (Yin, 2014) and best practice in system dynamics (e.g., parsimony). Hence, next to the notes taken during and the memories about the workshop, particularly the initial literature review served as a comparison to the empirical findings. The researcher presented and explained the refined models in the next sessions to the participants, requested them to deliberately challenge the model, discussed the implications with them, and adjusted the model according to the participants' comments, thereby increasing the models accuracy and the study's internal validity (Andersen et al., 2012; Vennix, 1996; Yin, 2014). As common in qualitative system dynamics research (see e.g., Repenning, & Sterman, 2002), the group model building sessions were later followed by further communication via e-mail, chat, phone calls, corridor conversations and also unstructured interviews.

Such further communication was deemed to be particularly useful because of the massive data constraints in the areas of cyber security and agile software development, arising from their respective nature: Security strives to overcome insecurity and uncertainty, and thus, there are limited data; agile software development (which is used in the organisation) is governed by flexible approaches with little documentation, and thus, leaves few reliable data behind. Consequently, next to group model building, the author had several informal conversations and eleven unstructured interviews, explicitly observed four times two DevOps teams, conducted further informal observations while being on side, and examined documents and archival data (Yin, 2014).[20] The researcher took notes about all activities as tape recording was not possible due to the security environment of the study and coded the data according to common practice in qualitative research (Merriam, 2009; see Appendix II, in the following only indicated by the number). The two DevOps team were observed during two short (around 20 minutes) and two longer (around 60 minutes) organisational meetings. The interviews were conducted according to the organisation's internal culture, meaning that they were scheduled and took place as usual work meetings. Due to the busy work environment, most of the participants were not asked to double check the researcher's notes. While this adaptation to the business environment reduced the validity of the findings, insights were anonymously discussed with different experts to offset that issue.

[20] Additionally, since internal documents are confidential they can neither be cited and referred to, nor made available to anybody outside the organisation. The researcher assures, however, that all documents and archival data were investigated by following academic standards.

More generally speaking, the collected data was continuously compared to the previous insights from literature and group model building, as common in qualitative research. Since the outcomes of the study will be employed within the organisation, the process of scrutinising the collected data and analysing it in the right context was strongly encouraged and supported by the manager of the responsible team. Consequently, all of the insights were constantly discussed with the gatekeeper, the responsible for software development and other experts to double check the accuracy of information and to examine alternative explanations. As such, contrasting empirical findings from single case studies with theory helps to increase the external validity by looking for generalisability. Discussions and analysis were once more guided by the aim of explaining the interaction between pressure in software development, software vulnerabilities, cyber attacks, and attack mitigation through the integration and comparison of literature and empirical insights (Merriam, 2009; Yin, 2014). Since phenomenon-based research and research in software engineering aim to be relevant for management practice (von Kogh et al. 2012; Sjøberg, Dybå, Anda, & Hannay, 2008), the outcomes were summarised in a comprehensive model. Discovered and discussed policy options for improvement were added as well. Finally, the findings were generalised and abstracted even further beyond the field of software development and cyber security to make a first step in building a dynamic theory of vulnerabilities. Overall, having triangulated the findings and employed a broad range of qualitative methods for data collection and data analysis increased the study's construct and internal validity (Thurmond, 2001; Yin, 2014). To further increase the internal validity, the qualitative models arising from the research were subject to explicit structure validation (e.g. Barlas, 1996; Forrester, & Senge, 1980) through disconfirmatory interviews with a specialised system architect, the responsible team, and the main responsible expert for secure software development (Andersen et al., 2012; see also II. E for further details). The approaches of data collection, the number of conducted activities, and the data analysis techniques were summarised in Table 1.

Table 1: Summary of Data Collection and Analysis

Data Collection	Number	Data Analysis
Group Model Building (GMB)	3	Group model building, iterative model building with explanation building and continuous comparison, discussions and disconfirmation
Notes about GMB	3	Coding, categorising, continuous comparison, explanation building
Unstructured Interviews	11	Coding, categorising, continuous comparison, explanation building, disconfirmation
Conversations	7	Coding, categorising, continuous comparison, explanation building
Observations	5	Coding, categorising, continuous comparison, explanation building
Documents, Archival Data	-	Informal review and discussion, confidential

3.4 Validity and Reliability

In system dynamics research and case study research validation is understood as a gradual, prolonged, and important process of iteratively and incrementally building confidence in the research and a model with each new insight, method, or test (Barlas, 1996; Forrester, & Senge, 1980; Yin, 2014). Although mentioned previously throughout the study, this subsection summarises the different approaches to ensure the model-based case study's validity and reliability. The case study ensured its construct validity (i.e., using the correct operational measures for the issue) by relying on multiple sources of evidence and data collection methods (I. B; II. C, D) and triangulating the insights (II. B). While notes about the data collection were not reviewed by the participants due to the business environment, all insights and the state of the research were continuously discussed with experts from the collaborating financial organisation. Finally, the gatekeeper and the main responsible for software security reviewed this study. Next, two approaches ensured the study's internal validity (i.e., establishing causal relationships): First, the empirical data was analysed through the technique of explanation building and causal diagrams were used as models (Yin, 2014). Second, the causal diagrams created throughout the workshops and later by the researcher and also the insights from the study as a whole were subject to critical assessment through three disconfirmatory interviews which aimed at refuting the findings as much as possible (see II. E for further details on disconfirmatory interviews). Furthermore, the study ensured external validity (i.e., assessing the generalisability of the findings) through constantly comparing the empirical insights with theory, and by explicitly generalising the findings in a first step of building a theory of the dynamics of vulnerabilities (5.2). Finally, the study's reliability (i.e., the study can be repeated) was ensured with a clear documentation of the entire case study, such as several versions of the causal diagrams (I. B), scripts of the group model building sessions (II. A), or coded interviews (II. D.; see for the entire documentation I and II). In addition, the study applied explicit techniques for model validation common in system dynamics research. Since the model developed for the purpose of this study is qualitative in nature, no quantitative validation (i.e., structure-behaviour-tests and behaviour tests) was applied. Instead the study relied on common techniques of structure validation in system dynamics research, namely structure verification, conceptual parameter verification, the discussion of extreme conditions with experts from the financial organisation, boundary adequacy, and unit consistency (i.e., in the built but not used quantitative model units are consistent) (Barlas, 1996; Forrester, & Senge, 1980).

3.5 Research Ethics

This study was conducted in collaboration with the cyber security department of a financial organisation in Europe. The collaboration inter alia provided the organisation with decision support to improve defence against cyber crime and malicious attacks. Thus, all participants in the research collaborated voluntarily. As this research was conducted in collaboration, the researcher had access to confidential information. To cause no damage to the collaborating organisation and its aims, the researcher handled all information with outmost care. Due to the security environment of the research, no meetings were tape recorded, and internal documents and archival data were not made publicly available but remained within the organisation as it is part of the agreement between Radboud University Nijmegen and the financial organisation. As part of the agreement, all confidential information that were supposed to be made public were double checked by employees of the cyber security department. The obligation to protect confidential information continues after the research has been terminated. Fully acknowledging the need to protect confidential data, the outcomes of the research are yet the intellectual property of the researcher.

The researcher followed other studies conducted at the intersection of software development, cyber security, organisational theory, strategy, and system dynamics. To this end, the researcher adhered to scientific standards and only applied suitable methods that he was able to conduct. The researcher gave credit to all sources, theories, ideas, and concepts used throughout this research. At the same time, the researcher refined the findings from literature, conducted own data collection and analysis, and by having combined the insights developed own concepts, answers, solutions, recommendations, and theories. Last but not least, the researcher declares that he has done all of the work independently and only used the declared and quoted sources. Passages in the text of this research which resemble other studies literally or in a general meaning were explicitly indicated as such through references and/or citation.

4. RESULTS AND ANALYSIS

Resources in software development are limited which forces DevOps to trade off between software functionality to address market pressure and software security to cope with malicious cyber attacks. Being subject to such tension causes pressure which is known to lead to mistakes (Austin, 2001; Rahmandad, & Repenning, 2016; Repenning, & Sterman, 2002; Rudolph, & Repenning, 2002). Since defects in software development become vulnerabilities upon release, an external malicious adversary may attempt to exploit such weaknesses created inside of an organisation. This results in even more pressure because the already limited time and resources have to be devoted to respond to an attack (Ahmad, Maynard, & Park, 2014; McGraw, 2006, 2012). Despite these dynamics, to the best of the researcher's knowledge, the described interplay has not been scrutinised in literature. Consequently, this study followed the objective to investigate and explain the dynamics of secure software development, software vulnerabilities, external cyber attacks, and organisational responses. Thus, the study addressed the following research question:

> *How does the interaction between pressure in software development, software vulnerabilities, external cyber attacks against an organisation, and the organisation's attempt to mitigate those attacks influence the trade-off between software functionality and software security?*

To approach the research question, this section presents, explains, and analyses a causal diagram which was developed by the researcher and derived from the empirical findings within the European financial organisation and the continuous comparison with the broader literature. The model unfolds in four steps, each representing one part of the investigated interaction between pressure in software development (4.1), software vulnerabilities (4.2), cyber attacks (4.3), and organisational attack mitigation (4.4).[21] This research follows the example of other studies in system dynamics (e.g., Repenning, & Sterman, 2002; Rudolph, & Repenning, 2002) by combining the presentation of findings from the model-based case study with explanations and analysis of the results. This approach provides the necessary context and relevance to understand the dynamic complexity governing the issue at hand. At the end of each step the major insights and their respective relevance are summarised in a table. Combined, the results provide the necessary conditions for answering the research question in the Discussion Section below (5.1).

4.1 Agile Software Development and Pressure

The first step was concerned with the interactions of planning and developing software and the possible occurrence of pressure, shown in the causal diagram of Figure 8.[22] Parts of the diagram were well known and very similar to Rahmandad and Repenning's description of software development (2016, p. 655-660). Other parts illustrated

[21] I. & II. show the iterative process of model building (Homer, 1996) and document the model, data collection and analysis.

[22] Causal diagrams (in system dynamics generally called causal loop diagrams) do not aim to provide mathematical descriptions of relationships to conduct simulation experiments as common in system dynamics research. Initially, the researcher had built such a mathematical model, but later not continued along that path. Next to severe data and time constraints, particularly the discovery of an anomaly in Rahmandad and Repenning's (2016) model which would have served as a starting point discouraged this endeavour.

the agile process of flexibly adjusting the workload which has not been modelled previously in software development but only in industrial push and pull production systems (Gonçalves, Hines, & Sterman, 2005).

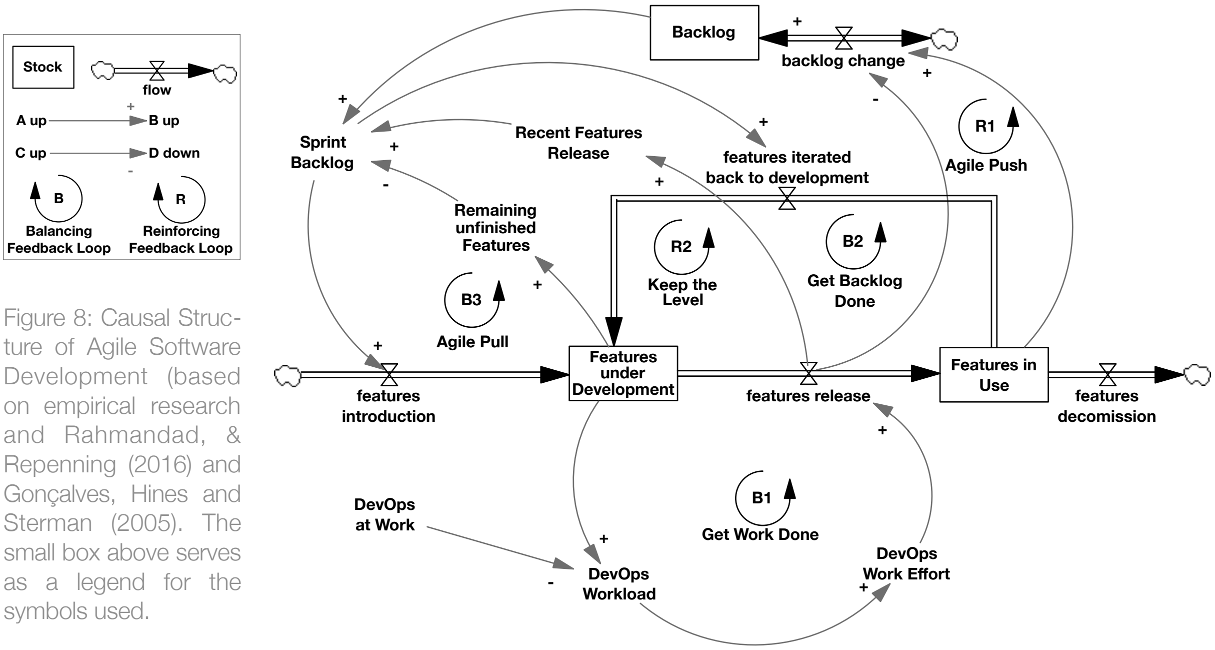


Figure 8: Causal Structure of Agile Software Development (based on empirical research and Rahmandad, & Repenning (2016) and Gonçalves, Hines and Sterman (2005). The small box above serves as a legend for the symbols used.

In contrast to sequential approaches in software development, theory on agile methods prescribes that external demand does not control the work of DevOps teams. Instead, the teams define the tasks set on the *Sprint Backlog* in collaboration with a representative from the business side, namely the so called product owner (Beck, et al, 2001; Pressman, 2010; Schwaber, 2004).[23] The Sprint Backlog lays out the work that a team plans to conduct per development cycle. The team selects the tasks from the product backlog and aims to deliver functional software at the end of the sprint. The product backlog, in Figure 8 simply called *Backlog*, describes the overall work regarding software throughout the entire lifecycle for all teams within the organisation (Pressman, 2010; Schwaber, 2004). Based on the Sprint Backlog, new features are introduced or existing features are brought back to development in an effort of continuous improvement, and together these build up in the stock of *Features under Development*. Stocks represent accumulations in a system and are changed through flows. As explained by Rudolph and colleagues, “stocks have a key role in creating dynamics: they create delays, give systems inertia, and provide systems with memory” (2009, p. 738). Although not displayed in detail here for the sake of clarity, the stock of Features under Development includes several phases of the lifecycle, namely, requirements, planning, design, development, testing, and pre-

[23] The names of variables that are shown in the model are italicised the first time they are introduced here.

paring for release. Once features are released they become part of the *Software in Use* within the organisation until they are decommissioned at the end of their lifecycle.

Based on the activities throughout development, release, and operations of software, DevOps teams plan the previously mentioned Sprint Backlog and select future tasks. In this context, the agile methodology borrowed several ideas from the lean approach in production to improve scheduling of tasks, empower employees, reduce unnecessary work, achieve fast delivery, understand value from the customer perspective, and optimise the overall product (Pressman, 2010; Widman, Hua, & Ross, 2010). The group model building workshops as well as observations of and interviews with two DevOps teams confirmed the similarities between agile software development and lean production. Amongst others, the empirical findings revealed that the planning of the Sprint Backlog follows the lean approach in the sense that it relies on the interplay of push, pull and anchoring mechanisms in production (Gonçalves et al., 2005). In practice, observations showed that the DevOps teams within the financial organisation arrange their future work based on what there is to do in total (Backlog), what they are used to do (*Recent Features Release*), and what they still have to do from the previous sprint (*Remaining Unfinished Features*) (II.D.12). Combined, these three techniques describe a dynamic interaction of push, anchoring, and pull mechanisms for determining the Sprint Backlog and play out as follows:

First, the Backlog functions as a push mechanism because DevOps feel urged to always handle it. The Backlog is increased by new ideas based on innovations in the market (not shown here for simplicity), and the overall number of Features in Use because these have to be maintained, operated, improved, or decommissioned. Hence, if the number of Features in Use is growing the overall work is rising. Assuming that the teams are able to fulfil the self-assigned tasks, the higher the Sprint Backlog, the more Features under Development, and eventually, the more Features in Use, leading to more work and closing the reinforcing feedback loop *R1 Agile Push*. Since a large part of the Features under Development is not introduced entirely new but iterated back from the existing features in an effort of continuous improvement, the overall number of features is only gradually increasing through this feedback loop. While R1 Agile Push increases the Backlog, the release of features decreases it, forming the balancing feedback loop *B2 Get Backlog Done* which counteracts the Backlog growth. Together, these two feedback loops determine the strength of the push-mechanism when planning the Sprint Backlog.

Second, the empirical data clearly showed that DevOps teams are oriented to their performance from the previous sprint which they implicitly use as an anchor for upcoming cycles. In this sense, the more features a team releases, the higher is the number of Recent Features Release, and thus, the higher the upcoming Sprint Backlog. Once more assuming that the teams are able to fulfil the assigned tasks, the higher the Sprint Backlog, the more Features under Development, and eventually, the more *Features Release*, closing the reinforcing feedback loop *R2 Keep the Level*. It is noteworthy that this feedback loop describes the mechanism of a floating goal (i.e., the desired state of a system is based on the current state of the system) as the DevOps teams anchor their future tasks on previous work. While this has the potential to drive performance, it can also lead to the erosion of goals if the team anchors on a constantly declining target (Sterman, 2000).

Finally, and potentially most importantly, the previous push and anchor mechanisms represented by the loops R1 Agile Push, B2 Get Backlog Done, and R2 Keep the Level are governed by unfinished work. Stated differently, the DevOps account for work in the stock of Features under Development which is still left from the previous sprint and only pull as many new features into the stock as can actually be developed and subsequently released within a sprint. According to the participants from the workshops, it is particularly the mechanism described by the balancing loop *B3 Agile Pull* that enables the DevOps to delay items on the Backlog and thereby prevent pressure occurring in agile approaches.

When confronted with the notion of stress and requested to comment on that, several participants in the workshops indignantly emphasised that there is no such thing as pressure in agile software development approaches. They added that within the organisation DevOps teams are fully self-organised and make use of the balancing pull mechanism of B3 Agile Pull, and thus, the teams are not subject to external pressure and stress from the product owner or higher management. One of participants commented:

1. *The agile approach clearly says that there is no pressure! (II.C.1 32)*[24]

In contrast to such claims, many conversations and observations within the six months on site created a more nuanced picture, underlining that pressure is indeed an present. According to a security expert for the topic of vulnerabilities, for instance, pressure is commonplace in software development because of the business perspective:

[24] The quotes are counted to refer back to them. Quotes are abbreviated with Q. The citation at the end of the quotation refers to the respective section and line in the Appendix.

> *2. The product owner from the business side looks first at functionality because that creates income, then at security because that costs money. We have enforceable standards in place, so people need to consider both, functionality and security. When there are deadlines this puts them under stress.* (II.D.4 75ff.)

Along the same line, release pressure was clearly visible among the observed DevOps teams. Having visited them at the very beginning of a sprint showed relaxed and calm people who had time for chats and who enjoyed joking around, whereas attending meetings with the very same employees at the second half of a sprint revealed higher stress levels within some of the teams, for instance, visible by people eating at their workspace, starting and finishing meetings exactly on time, admitting quite a high workload with a grim smile, or being curt and brief (II.D.9-12, 19). Similarly, one of the DevOps admitted previous very high workloads which he hopes to not get back to. Additionally, one of the managers in the security field strongly emphasised within a conversation that the organisation is slim and efficient, and that people are working to capacity:

> *3. We are a business, we have no unnecessary slack in this organisation and people are not just sitting around and waiting to do something!* (II.D.3 5ff.)

In short, the empirical data confirmed theory that teams are actually self-organised and plan their work in collaboration with the business side. However, the insights gained within the financial organisation also contradicted theory in the sense that also agile software development approaches are subject to considerable pressure. In this context, the Features under Development indicate the amount of tasks that *DevOps at Work* need to address, and combined, these two describe the *DevOps Workload* which effectively represents the pressure resting on the DevOps teams. Generally speaking, there are four options available to an organisation to address growing pressure: First, decrease the tasks on the Sprint Backlog, second, increase the number of DevOps at Work, third, increase the workweek, and fourth, decrease the time per feature (Sterman, 2000). Since market pressure is high in the software business, cutting on demand or delaying release is seldom considered as an option in many businesses because of the high short-term costs, even though such a postponement may save money in the long-term (Arora et al., 2006; Rahmandad, & Repenning, 2016). However, in contrast to software vendors, financial organisations do not conduct software engineering as their core activity. Customers appear to appreciate new features in software, but there is no evidence that they would prefer new features over properly functioning software. One of the security experts explained this situation:

> *4. Since we are a financial organisation time to market pressure is less of a problem for us. We are not an app developer. We do not lose market share if we release something later. We also do not have to address all customer demands, but instead make sure that the software that customers use actually functions in a proper way.* (II.D.21 23ff.)

At the same time, several other experts emphasised the business impact of using the possibility of delaying tasks on the Backlog too long.

> *5. The workload of a DevOps team should always stay stable. The problem is when it actually stays stable and you do not develop anymore because of all the other work you have to do. Then you delay the stuff on your backlog.* (II.D.6 335ff.)
>
> *6. Agile is fine with delaying work and that is a good thing to prevent pressure. But if you do that for too long, you create a strategic delay which may cost you, depending on the industry, several percent of your revenues.* (II.D.7 71ff.)

Hence, decreasing or halting the Sprint Backlog by pulling less features is a common option to decrease pressure. At the same time, continuously relying on it may create pressure from another angle as revenue losses start to occur in the long-term. Additionally, employees indicated the concern that people may become so used to delaying tasks that they implicitly account for it from the very beginning when planning future work. In other words, delaying tasks may not be an option in case of pressure because it is already used on a daily basis. Regarding the second option to adjust the overall workload, changing the number of DevOps is often not helpful in decreasing pressure as found out at IBM because "the greater costs of coordinating among more developers may outweigh any gain in efficiency" (Arora et al., 2006, p. 465, referring to the study of Brooks, 1995). To account for this difficulty, teams within the financial organisation are adjusted differently, as explained within the workshops and illustrated by one employee:

> *7. If we want to increase the people working on something, we do not increase the team size, but add new teams. Let's say we have a team of eight people. We then split the team in two teams with four people each and add four new DevOps to each team, and if necessary, we just continue like this.* (II.D.22 26ff.)

According to employees within the organisation this mechanism is not used for minor issues because of the costs involved but mainly when stress is very high and "you want to cut the pressure loop" (II.D.23 148). Additionally, an organisation need to have enough DevOps to rely on this approach. At the same time, since the last financial crisis starting in 2007, many financial organisations have been restructuring their staff which effectively means they have been laying off employees because of financial constraints and the growing importance of ICT (Crowe, 2016; Lopez, 2013; Rankin, 2013), also underlined by one employee:

> *8. Instead of hiring more DevOps for actually doing more work, they hired somebody basically holding a whip for making the DevOps work faster and to increase the pressure to deliver.* (II.D.18 39ff.)

Hence, to address pressure in the short-run, financial organisations often prefer to make their employees working harder by increasing the workweek and decrease the time spent per features, summed up in Figure 8 above as *DevOps Work Effort* (Oliva, & Sterman, 2001; Sterman, 2000). Increasing the work effort of DevOps boosts the release of features and consequently decreases the Features under Development which effectively reduces the workload of a sprint. In sum, these causal links represent the balancing feedback loop *B1 Get Work Done* which basically describes the work effort of DevOps to process all tasks selected for the Sprint Backlog, if necessary by working more or spending less time per task. As pointed out by Rahmandad and Repenning, "this loop gives a team a significant measure of flexibility in managing the inevitable variations in its workload" (2016, p. 657).

Table 2: Summary of Results in 4.1 Agile Software Development and Pressure

Findings	Empirical Evidence*	Relevance
DevOps in agile software development select their tasks for the sprint backlog based on push, anchor and pull mechanisms known from lean production.	Q 1 II.D.12	Anchoring in agile methods may function as a floating goal, enabling high performance or eroding goals. The pull mechanism is the safeguard against pressure.

Despite the self-organised nature of planning, DevOps in agile approaches are under significant pressure.	GMB;
	Q 1, 2, 3;
	II. D. 9, 19
		Theory about agile software development approaches clearly emphasises that no pressure should arise. Since it is known that pressure decreases the quality of the product, agile methods may not be able to hold the promise of quality and customer orientation.

Financial organisations become "technology companies with a banking licence". Yet they are not subject to the same market pressure as software vendors.	Q 4	For financial (and other non ICT-) organisations, customers are customers because of the product and less because of the time to the next feature release.
To overcome pressure, tasks may be delayed which potentially causes a strategic delay, leading to heavy cuts in revenues.	GMB Q 5, 6	A potential downside of agile approaches is the ease of delaying tasks which decreases pressure in the short term but may heavily increase it in the long-term.
To overcome pressure, DevOps teams may be split and filled up with new DevOps to avoid the coordination problems of managing large teams.	Q 7	Splitting the DevOps teams and the tasks enables an organisation to address pressure by shifting DevOps. While an organisation needs to have enough DevOps at hand, this approach may serve as a powerful mechanism to "cut" pressure loops.
Most commonly, to counter pressure, work effort is increased.	GMB Q 8	DevOps work either overtime or spend less time per task.

* Empirical evidences may refer to the appendix (e.g., to broader observations, such as II.D.12), to group model building workshops (abbreviated with GMB) or to quotations from the text presented above (abbreviated Q). As previously mentioned, the term appendix is not written in sources but just the respective section and subsection is given (i.e., I.A, B, C, D, E, F; II. A, B, C, D, E) In addition, sometimes reference to other studies is given, such as in Table 3 below, referring to Rahmandad & Repenning, 2016.

4.2 Defects and Vulnerabilities

Having said that, Oliva and Sterman (2011) found out that employees within banks are less likely to increase their workweek, but instead take shortcuts to meet demand because they do not want to work longer. Decreasing the time per task opens a path to sacrificing quality against meeting pressure though, because the needed discipline and steps within the software development lifecycle are omitted (MacCormack et al., 2003; Rahmandad, & Repenning, 2016; Pressman, 2010). Hence, next to the inevitable number of defects that simply occur in software development, the higher the work effort of DevOps, the more additional defects are introduced into a software (Figure 9). While literature distinguishes between many different classifications of defects,[25] this study only differed between unknown and known defects. Additionally, the causal diagram in Figure 9 combines both kinds of defects, functionality and security, to decrease the complexity of the diagram. Increasing DevOps Work Effort causes a growing stock of *Unknown Defects*. Since it is assumed that DevOps do not introduce defects intentionally,[26] defects are obviously

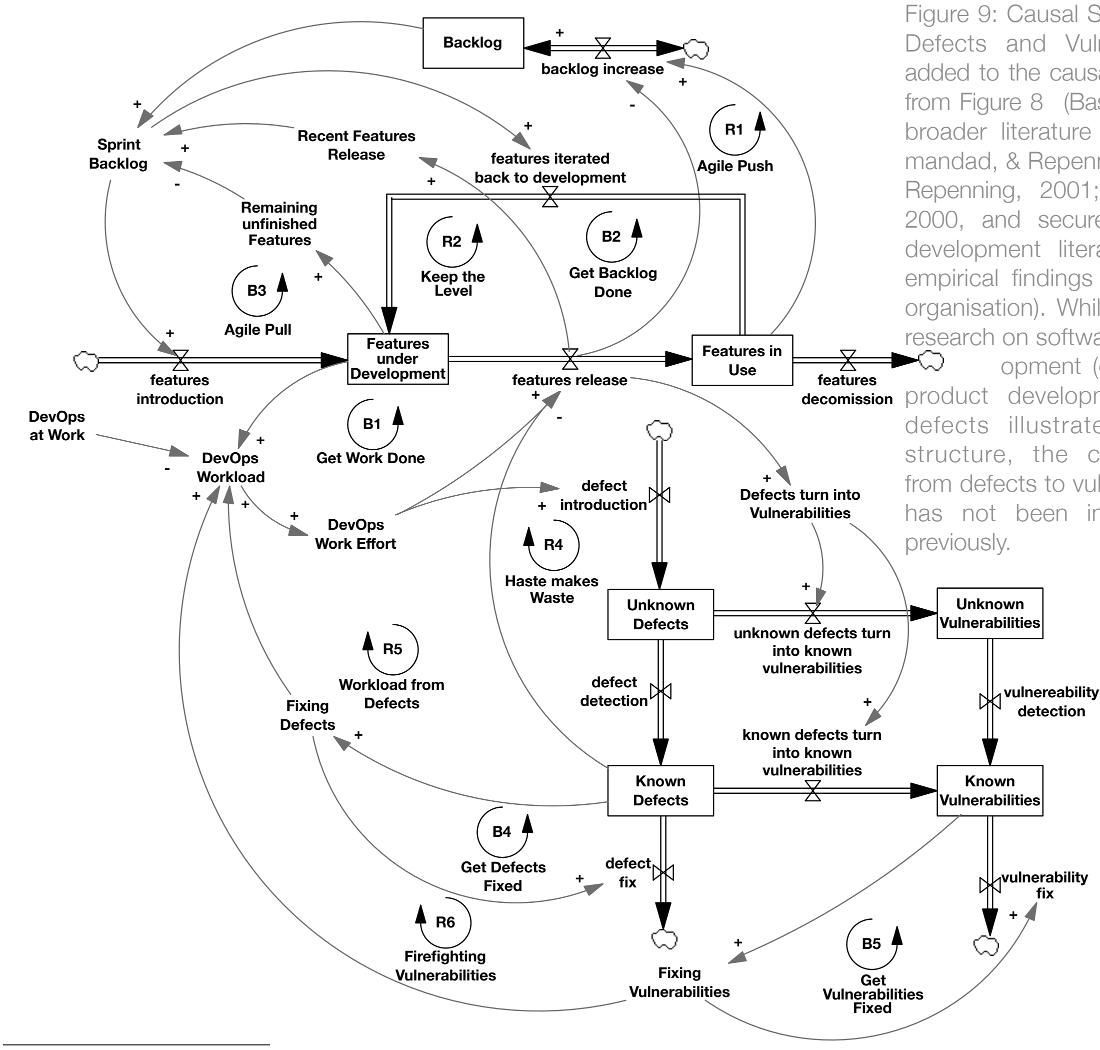


Figure 9: Causal Structure of Defects and Vulnerabilities, added to the causal structure from Figure 8 (Based on the broader literature incl. Rahmandad, & Repenning, 2016; Repenning, 2001; Sterman, 2000, and secure software development literature, and empirical findings within the organisation). While previous research on software development (or broader product development) and defects illustrated similar structure, the connection from defects to vulnerabilities has not been investigated previously.

[25] McGraw (2006), for instance, distinguishes flaws (i.e., defects in design) and bugs (i.e., defects in implementation).

[26] As previously mentioned, this study does not account for insider threats. See for this topic e.g. Martinez-Moyano et al., 2008.

unknown and need to be detected through testing. While testing used to be modelled explicitly in the system dynamics literature (see e.g., Rahmandad, 2005), nowadays organisation commonly rely to a very large extent on automated solutions which increase the effectiveness of tests and drastically decrease the required work for DevOps. One of the observed and interviewed DevOps teams emphasised and discussed this benefit:

> *9. One of us used to spend his entire time on testing the developed software. Now with automation, we have much more time for developing and operating. Of course, we still do manual testing but by far not as much as we used to.* (II.D.19 77ff.)

Consequently, DevOps are still spending some time on testing but this is limited enough that it is not considered anymore in the aggregated model depicted in Figure 9, but just understood as part of the overall development work. It is noteworthy though that automation leads to the discovery of all defects. In contrast, an external experts referred to the growing number of defects in the world:

> *10. The global errors in software are piling up. This is potentially not recognised but the overall number is, in my opinion, by far larger than actually reported.* (II.D.14 13ff.)

Additionally, there is a misperception about what automation can actually do. In a team meeting, two experts explained the rest of the team and the leading manager the limits of this technology as it was perceived that more than mere testing should be possible:

> *11. Automation is simply for testing. Yes, there are a few other things we can do with it, but for now this is limited. It saves us lots of time with testing and gives better findings than many manual tests, but we still need to fix it ourselves.* (II.D.17 46ff.)

In a private conversation, an employee within the financial organisation made his view of the problem of the current way of employing automation clear.

> *12. Finance basically says 'you get automation, but how many people do we save by that?' The problem there is, even with automation we do not get better. It is true that we detect more but since we have fewer people, we can't benefit from our increased knowledge because we can't fix it.* (II.D.18 22ff.)

In short, automation helps to decrease the workload for testing and to increase the *defect detection* but it does not improve the workload when addressing defects.

Once defects are detected, they accumulate in the stock of *Known Defects*. Since theoretically these defects are not supposed to be in a software upon release, they lower the release rate which unintentionally increases the stock of Features under Development, causing more pressure, eventually leading to more defects, thereby building the reinforcing feedback loop *R4 Haste makes Waste*. Additionally, *Fixing Defects* prior to release decreases the number of Known Defects (*B4 Get Defects Fixed*) but immediately increases the DevOps Workload because DevOps have to flexibly allocate their time to resolve the defects. Since this causes even more pressure which results in more defects, there is the potential that DevOps get caught in the reinforcing feedback loop *R5 Workload from Defects*. Consistent with the Yerkes-Dodson Law and previous studies (e.g., Burchill, & Fine, 1997; Rahmandad, & Repenning, 2016; Repenning, & Sterman, 2002; Rudolph, & Repenning, 2002), high pressure creates more work and results in a vicious circle, while lower pressure means introducing fewer defects which results in a virtuous circle. As pointed out by Rahmandad and Repenning (2016), the vicious circle may become that strong that the benefits of increased work effort do not compensate for the defects anymore, but cause software development and operations capabilities to erode. While the potential of this capability erosion due to reinforcing defect pressure has been confirmed by the responsible security expert for software development within the financial organisation when confronted with the u-shaped curve of the Yerkes-Dodson Law, the expert also pointed out that Fixing Defects prior to release is common in agile software development and does not necessarily constitute a clear sign of problems and stress (II.D.23).

Additionally, organisations may simply decrease pressure by recognising problems and fixing them later after release, commonly described as the mentality of “sell first, fix later” (Arora et al., 2006, p. 465). While market pressure initially caused stress, the decision to release first and fix later actually decreases the immediate stress to solve defects prior to release and allows organisations to address them later when time and resources are available. In other words, DevOps teams have the opportunity to decide whether and when to fix defects, thereby decreasing stress from rework and reducing future defects due to pressure. Although the observed and interviewed DevOps teams in the financial organisation clarified amongst each other that software is not released as long as Known Defects are present (II.D.10), it is common practice in many industries, that both kinds of defects, functionality and security, are frequently not solved before release. Having said that, primarily security issues remain in software upon release due to the business-driven perspective of software development, as pointed out by several employees within the organisation:

> *13. Particularly in agile, software is developed for functionality because of the customer-oriented approach and security is then seen as an add-on.* (II.C.1 35ff.)

> *14. The desired software behaviour for people from operations and business is not security. They do not care because security doesn't give money and if nothing happens nobody even recognises the success of security because you don't know why nothing happened.* (II.D.2 20ff.)
>
> *15. DevOps often lose the fight against the product owner who prefers functionality over security.* (II.D.15 90f.)
>
> *16. When developing functionality, we know the use cases. Security is more difficult because we must think of abuse cases. When we need to decide, often the certain use cases come first, and then the uncertain abuse cases.* (II.D.20 81ff.)
>
> *17. Functionality brings the team more prestige here within the organisation. Nobody sees security. So, they go for functionality and for the prestige.* (II.D.20 70ff.)
>
> *18. Long-term benefits from security are sacrificed for short-term gains from functionality because people think it is so unlikely that something happens.* (II.D.23 104ff.)
>
> *19. If there is high business pressure, you go for functionality because you need to survive business. You should not do that for too long though but most of the time you would.* (II.D.21 58ff.)

Particularly the last comment underlined the common habit in software development to trade off security for functionality due to continuously high pressure from the business side. While potentially less dramatic than with commercial software vendors or mobile app developers due to less "time to market" pressure in the financial industry (Q 4), the empirical data indicated that the business-driven perspective and pressure from competition in the financial industry do still invite to cut corners, causing defects which exacerbate stress, reduce quality and security, and possibly introducing a vicious circle of developing fast, releasing fast, and fixing a lot.

These interactions alone may not be problematic because developing fast and Fixing Defects is common practice in agile approaches as described by the expert above. However, three issues arrive from this practice: First, the later software defects are fixed, the more expensive they are (e.g., Boehm, 1984). It is cheaper to not introduce a defect at all, than fixing it within the same sprint, or three sprints later, and so on. Second, the older a defect becomes and the more features are based on flawed previous work, the more difficult it is to resolve, a problem referred to as "technical debt". To avoid this problem, agile approaches employ an activity called refactoring to "reduce software complexity by incrementally improving internal software quality" (Cao et al., 2010, p. 5). A security expert who used to work as a DevOps elaborated on the theory and practice of refactoring:

> *20. Refactoring should be done on a regular basis as part of the normal development process. Sadly, in so many companies it is never done.* (II.D.15 122ff.)

Interestingly, the obvious need to regularly conduct refactoring is not shared by all involved parties in the development process, as for instance indicated by the representative from business in one of the DevOps teams:

> *21. Why should we take the time for refactoring? It is obviously done enough because otherwise it would be a priority on our Sprint Backlog. (II.D.19 102ff.)*

This argument is noteworthy because it underlines the sacrifice of the long-term quality and security for short-term gains from the fast release of software, and because it shows that the judgment of DevOps teams may be biased. Since the risks for quality and security from complex software are difficult to imagine, DevOps may underestimate the potential future problems arising from it, a mechanism commonly referred to as the availability heuristic discussed by Tversky and Kahneman (1974). Third, and in the context of this study most important, the practice of developing fast and Fixing Defects, leading to the reinforcing feedback loop R5 Workload from Defects, has severe security implications. Once released, *Defects turn into Vulnerabilities* as shown in Figure 9 above. Obviously, Unknown Defects and Known Defects accumulate in the stocks of *Unknown Vulnerabilities* and *Known Vulnerabilities* respectively. Considering Figure 2, it is noteworthy that up to fifty percent of vulnerabilities are unknown. While DevOps are not involved in v*ulnerability detection*, particularly the activity of *Fixing Vulnerabilities* has the potential to heavily disrupt their regular work. Following the same mechanism as described above for defects, DevOps allocate their time to fix vulnerabilities (*B5 Get Vulnerabilities Fixed*) which simultaneously leads to more workload. While defects are common in software development, empirical data indicates that DevOps teams do not expect their products to be vulnerable:

> *22. We would know whether the features are vulnerable because pentest would tell us. Since that has not been the case, there are no vulnerabilities. (II.D.19 122ff.)*

While it is true that penetration testing is a powerful detection mechanism (Arkin, Stender, & McGraw, 2005), findings from other detection techniques within the organisation indicate that software vulnerabilities do occur despite having been subject to penetration testing. Underestimating the probability of vulnerabilities (Tversky, & Kahneman, 1974), DevOps may become subject to unexpected pressure when vulnerabilities are detected and the activity of Fixing Vulnerabilities disrupts their planned work in a sprint. Such a disturbance decreases the Features Release, leading to more Features under Development which, in turn, increases the DevOps Workload even further. This results in more defects and subsequent vulnerabilities which have to be fixed as well, causing even more work. As a consequence, DevOps teams may fall into the vicious circle *R6 Firefighting Vulnerabilities*, in which they mainly focus on fixing vulnerabilities instead of releasing software. This mechanism, particularly investigated by Repenning, describes a dangerous downward spiral process “whereby lack of attention to the early phases of the development process results in serious problems when projects reach their downstream phases” (Repenning, 2003, p. 305). Since time and software complexity play a significant role in determining the ease of fixing a defect or vulnerability, the lack of attention to upstream activities (e.g., design or misuse cases) creates problems and pressure once it comes to the downstream activity of Fixing Vulnerabilities. Along the same line with studies from Repenning and his colleagues (see e.g., Rahmandad, & Repenning, 2016; Repenning, 2001, 2003; Repenning, Gonçalves, and Black, 2001; Repenning, & Sterman, 2002), such a focus on downstream activities may trap the DevOps in a downward spiral of firefighting activities, constantly eroding performance. The responsible security expert for software development emphasised the difference between pressure from defect fixing and the pressure from firefighting vulnerabilities:

> *23. Both are plausible, both may lead to problems, but while the first is normal in agile methods, the latter should not occur.* (II.D.23 125ff.)

Consistent with the empirical findings of this study, Rahmandad and Repenning (2016) explain a similar mechanism in software development which focuses on current engineering in the case of a software vendor. Current engineering describes the activity of fixing significant defects at the customer side once those errors are detected after release (Rahmandad, & Repenning, 2016). As explained by the authors, the “well-intentioned efforts by managers to search locally for the optimal workload balance lead them to systematically overload their organization and, thereby, cause capabilities to erode” (Rahmandad, & Repenning, 2016, p. 649). Due to the time delay between releasing the software and detecting the defect, the organisation allocates resources to development activities and does not expect to need the very same resources for future defect fixing. As a consequence, “in the short run, the system will behave as though it has more capacity than is actually available” (Figure 10) (Rahmandad, & Repenning, 2016, p. 661). As suggested by the authors, their theory called the Adaptation Trap is applicable in other settings as well, such as the one described in this study: Despite applying agile software development approaches and relying on

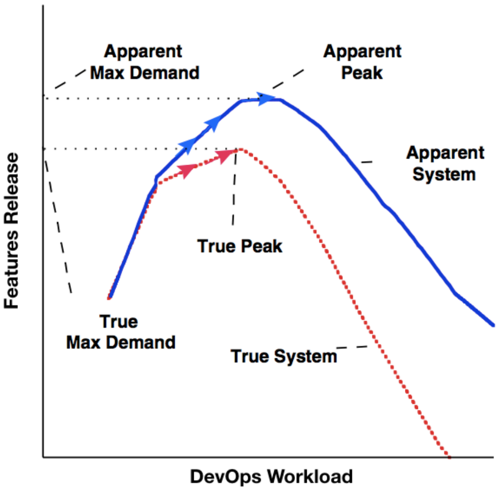


Figure 10: Adaptation Trap (Rahmandad, & Repenning, 2016, p. 661). The blue line indicates the apparent, the red line the real relationship between pressure (name of x-axes adjusted to DevOps Workload, in original "Resource Ratio") and performance (Features Release).

new technological possibilities, such as automated testing, pressure causes DevOps to increase their work effort which boosts the release of features but also results in defects. While it is never possible to detect and fix all errors in a software and particularly pressure prevents DevOps from addressing all Known Defects, the business-driven perspective results in neglecting security defects. Hence, vulnerabilities arise upon release. While teams can plan their sprint with Known Vulnerabilities, Unknown Vulnerabilities are detected with an average delay of about month (Ablon, & Bogart, 2017). Depending on the number of newly detected vulnerabilities after release, DevOps may not have enough capacity to actually cover their Sprint Backlog and the newly arisen vulnerabilities. Eventually, this wrong adaptation may lead to high pressure and continuous firefighting, constantly eroding the capability to develop and operate software, causing harm to an organisation's performance and success.

Table 3: Summary of Results in 4.2 Defects and Vulnerabilities

Findings	Empirical Evidence	Relevance
Automated testing improves the efficiency and effectiveness of defect detection but does not help to fix the found defects.	Q 9, 10, 11, 12, 22	To reap the benefits of automated testing, enough DevOps have to be available for fixing the detected defects. Else, the investment in automation is likely to have little positive effect.
In agile approaches, fixing defects prior to release is not necessarily a sign for pressure or bad quality.	II. D. 23	The acknowledged effects of R4 and R5 in literature also exist in in agile approaches but may be much less problematic.
The "sell first, fix later" mentality functions as a mechanism to decrease pressure.	Arora et al., 2006	While the decreased pressure reduces the number of future defects, "sell first, fix later" also allows vulnerabilities to arise, probably creating new pressure.
Due to the business-driven perspective and pressure resting on the DevOps, long-term benefits from security are regularly sacrificed for short-term gains from functionality.	GMB Q 13, 14, 15, 16, 17, 18, 19	In contrast to common practice, business risks must be valued against security risks, and only thereafter, should a decision be taken about sacrificing long-term robustness and quality.
DevOps teams are subject to availability heuristics.	Q 21, 22	Due to the difficulty of imagining complex software problems in the future or successful cyber attacks, the problem of vulnerabilities is unintentionally played down.
Fixing vulnerabilities may catch DevOps in a persistent firefighting mechanism, particularly if large amounts of unexpected work arise.	GMB Q 23; Rahmandad & Repenning, 2016	Wrongly adapting to the actual workload in the future by not accounting for the need to fix vulnerabilities may lead to permanent capability erosion.

4.3 Adversary Dynamics

Rahmandad and Repenning (2016) indicated as one of the major safeguards against wrong adaptation a dedicated learning approach in which teams would take time to learn from past errors. While acknowledged among the DevOps, this activity is often not done:

> *24. I'm new here, so I was really wondering… Why do we not grab the coder and do the code review together. I mean, it's a great learning opportunity, we should really consider doing that, but I guess we don't have the time, right?* (II.D.11 52ff.)

More importantly, Rahmandad and Repenning (2016) suggested a fixed resource allocation to the activities of fixing unexpected and unknown work. More precisely, in their investigated organisation ten percent of the developers were allocated to current engineering. If there was less to do the developers had no work, and if there was more to do the work had to wait. Consistent with literature on solving dynamic problems, the authors suggested an approach of stepping back, taking time, and reframing the situation in order to address the issue. Similar to the insights gained from Rudolph and Repenning's (2002) study on disaster dynamics, in the case of software vulnerabilities this approach of wait, learn, and see may be counterproductive as emphasised by several security experts:

> *25. Vulnerabilities can be exploited immediately, so you need to fix fast.* (II.D.6 227f.)
>
> *26. We need to have a very short mean time to resolve to reduce the risk.* (II.C.2 137)

Hence, the total number of Unknown Vulnerabilities and Known Vulnerabilities increases the *Probability of a Successful Cyber Attack* as depicted in Figure 11.

The Probability of a Successful Cyber Attack is affected in several ways: First, since attackers employ the same tools as an organisation when searching for defects and vulnerabilities, vulnerabilities known to an organisation are often easy to find for external attackers as well and may be exploited fast. Second, Unknown Vulnerabilities have to be detected prior to an exploitation which takes on average one month, indicated by the double line crossing a link in the diagram above. Depending on the vulnerability, new techniques for conducting the attack have to be created which takes a further three weeks on average (Ablon, & Bogart, 2017). While it takes time for an attacker to address Unknown Vulnerabilities, these so called *Zero Day Exploits* are very powerful as organisations have no defence mechanisms in place to protect themselves.[27] Third, the danger arising from the total number of vulnerabilities accumulated in both, the stocks of unknown and known vulnerabilities, is moderated by their criticality, namely low, medium,

[27] More formally, "*Zero-day vulnerabilities* are vulnerabilities for which no patch or fix has been publicly released. Thee term *zero- day* refers to the number of days a software vendor has known about the vulnerability" (Ablon, & Bogart, 2017, p. iii).

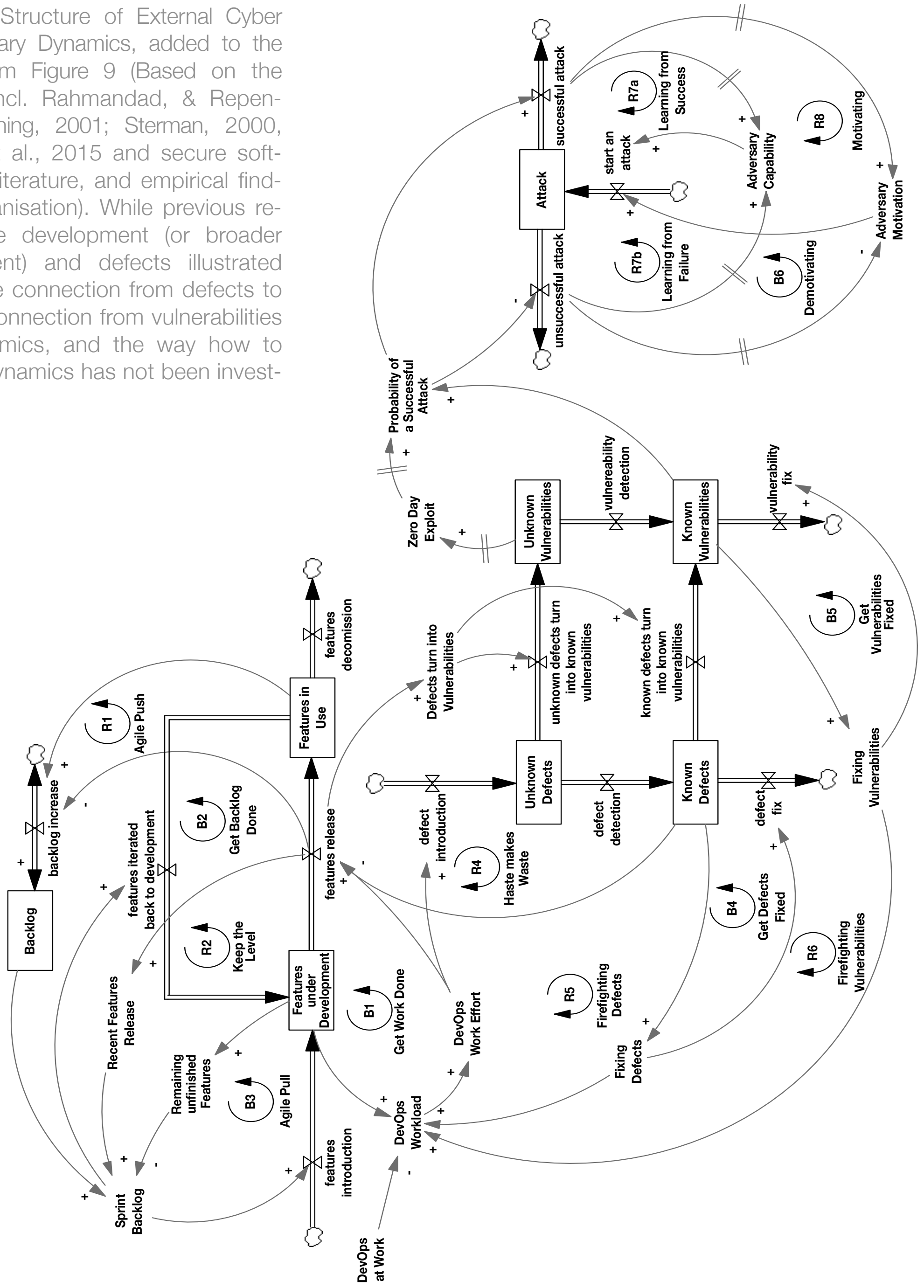


Figure 11: Causal Structure of External Cyber Attacks and Adversary Dynamics, added to the causal structure from Figure 9 (Based on the broader literature incl. Rahmandad, & Repenning, 2016; Repenning, 2001; Sterman, 2000, Martinez-Moyano et al., 2015 and secure software development literature, and empirical findings within the organisation). While previous research on software development (or broader product development) and defects illustrated similar structure, the connection from defects to vulnerabilities, the connection from vulnerabilities to adversarial dynamics, and the way how to depict adversarial dynamics has not been investigated previously.

high, and critical. According to security experts in the organisation (II.C.2; II.D.4,6,15), companies across most sectors only fix critical vulnerabilities immediately. Other vulnerabilities are added as tasks to the Backlog in order to be fixed later to not disrupt the regular activities. A security expert pointed out the problem with this technique:

> *27. People often say 'we do it later because now we really do not have the time', but then later it is simply not put on the Sprint Backlog and just not done. (II.D.15 129ff.)*

Hence, critical vulnerabilities are generally addressed fast because of their great danger for an organisation, whereas vulnerabilities of lower criticality are considered much less, enhancing the Probability of a Successful Cyber Attack (see also II. D. 6). While it makes perfect sense from a business perspective to only disrupt the process of developing and operating software for critical vulnerabilities to optimise value, the aspect of quantity is of high relevance in this context. The same security expert further elaborated:

> *28. Vulnerabilities are not only critical because of their level, but also because of the underlying mathematics. The numbers count because many low and medium vulnerabilities may be as dangerous as one or two critical ones.* (II.D.15 41ff.)

Consequently, the Probability of a Successful Cyber Attack is influenced by the quantity, the criticality, and the state of the vulnerability, namely whether it is known or unknown. Omitting one of these characteristics (as often done with lower criticalities and the quantity of those) unintentionally increases the Probability of a Successful Cyber Attack. The higher this probability, the more *successful attacks* and the *less unsuccessful attacks*.

According to Martinez-Moyano and colleagues (2015), attacks may generally be displayed as projects under development. Along the same line, the participants from the group model workshops (II.C.2) and two experts from within the financial organisation, one an ethical hacker, the other a software security expert (II.D.1, 2), described how hacking attacks evolve in three distinct steps: First, adversaries search for information and scan all external facing ICT of an organisation and may also target employees to receive information. Next, the gained insights are used to search for vulnerabilities. Finally, in case of detected vulnerabilities the adversary searches for a way to exploit them. If such a utilisation of an organisation's weaknesses is possible, a successful attack takes place. If the exploitation is not possible, the adversary may continue the search or end the project, leading to an unsuccessful attack. While displayed in detail within the group model building workshops (I. B), Figure 11 simply incorporates all these activities within the stock of *Attack*. Next to hacking attacks, also malware exploits software vulnerabilities, such as in the recent cases of WannaCry and NotPetya (Fox-Brewster, 2017). While very different regarding the distinct steps and often by far not as sophisticated as hacking attacks, also malware-based attacks follow the logic of a project under development because malware needs to be developed for a certain purpose, pass different levels of security layers within an organisation, reach its intended destination, and finally execute its purpose.[28]

Next to the Probability of a Successful Attack, mainly the *Adversary Capability* and the *Adversary Motivation* determine the initiation and later the outcome of an attack (II.C.2, 3; II. D). While the causal diagram above only links Adversary Capability and Adversary Motivation to *start an attack* for simplicity reasons, of course, their intensity defines the

[28] Prior to this study, the author and two colleagues have led workshops in the same financial organisation on malware-based attacks.

actual strength of an attack. Simply speaking, the more effective and the more motivated an attacker, the higher the chances of successfully exploiting a software vulnerability and having a successful attack. Adversary Capability sums up the effectiveness of an attacker, and describes the routinely employed skills of an adversary to use the available resources. Interestingly, participants in the workshops and further security experts pointed towards the similarity between cyber adversaries and business organisations.

> *29. Attackers are not different from companies, they have objectives, teams, maturity, tools, and so on. They are just using different, and illegal methods.* (II.C.2 161ff.)
>
> *30. An attacker has a business case, like we have one too.* (II.D.6 396)

Hence, also cyber adversaries form an objective and develop strategies, acquire resources, and strengthen capabilities to increase performance and achieve success. In the context of an adversary, resources may describe money, ICT, people or anything else that is necessary for an attack. Skills include the ability to develop malware or to conduct a hacking attack. In contrast to physical attacks, such as in terrorism (Martinez-Moyano et al., 2015), cyber attacks may be successful despite very limited resources or skills. Adversaries can buy cheap exploitation tools, malware, relevant information or even attack services on the black market (Libicki, Ablon, & Webb, 2015), and often malware attacks are sent randomly to many targets. Success in the latter case does not need a sophisticated attack but rather a defender not being aware of cyber security. According to the participants, adversaries increase their capabilities from both unsuccessful and successful attacks. In most cases, resources are not negatively affected by conducting an attack (i.e., malware is not depleted like ammunition after use),[29] and successful attacks may lead to new information or actual monetary value. Since cyber adversaries are generally strategically thinking and creative actors, skills generally increase by any outcome of an attack because both, successful and unsuccessful attacks, offer learning opportunities (e.g., Libicki, Ablon, & Webb, 2015; McGraw, 2006). Hence, growing numbers of conducted Attacks offer more learning opportunities which increase the Adversary Capability, thereby creating the two reinforcing feedback loops *R7a Learning from Success* and *R7b Learning from Failure*. Depending on the insights gained from an attack, an adversary may repeat the very same attack or displace the criminal activities (II.C.2), namely to another time (e.g., night, public holidays, or after publicly announced software release), place (e.g., web-

[29] For malware specifically developed for exploiting a zero day vulnerability (e.g., Stuxnet, see Miller & Rowe, 2012) this is not true. The resource is lost because it derives its high value from being a possibility to exploit an Unknown Vulnerability which is known afterwards.

page or mobile app), method/tactic (e.g., malware or hacking), target (e.g., another organisation), or offences (e.g., theft or blackmail). As described in the crime displacement literature, these displacement techniques may occur simultaneously to improve Adversary Capability and achieve the initial objective (Hesseling, 1994; Johnson, Guerette, & Bowers, 2014; Telep, Weisburd, Gill, Vitter, & Teichman, 2014). Combined, the two reinforcing feedback loops describe a potential pathway to escalation from learning and practice since an adversary strengthens his/her own skills by any outcome of an attack. In practice, this escalation mechanism is moderated by forgetting rates, limits to learning, the number of exploitable software vulnerabilities, cyber security mechanisms, and particularly the Adversary Motivation. In other words, no matter how extensive the skills and resources of an adversary are, if there is no motivation to attack a specific target, no attack takes place (II.C.2). Further to the previously described displacement decisions to improve Adversary Capability, the initial objective and motivation has a particular impact on the decision to change or stay with a target, as explained by two security expert:

> *31. Generally, an attacker changes his target after an unsuccessful attack. If the attacker has a specific objective and conducts a targeted attack against an organisation, he will stay with that target because he has more information about it and he has a reason to attack that target.* (II.D.6 244ff.)
> *32. The escalation depends on the attacker and his target.* (II.D.23 131f.)

Hence, adversaries stay with a target in case of a successful attack, increasing Adversary Motivation, thereby forming the reinforcing feedback loop *R8 Motivating*. In combination with the two previous capability loops, this loop further escalates the activities of an adversary. If attacks are unsuccessful though, most adversaries would lose motivation and change their target, thereby decreasing the numbers of attacks against the same organisation, forming the balancing feedback loop *B6 Demotivating*. Since private organisations are not allowed to chase adversaries, only governmental activities, such as arresting attackers, may have further demotivating effects. Overall, the behaviour of cyber adversaries is driven by escalatory patterns and only unsuccessful attacks, government interaction, or the deliberate choice of another target due to an adversary's strategy decrease the overall number of attacks against a particular organisation. As indicated above, the strength of this balancing effect is further moderated by the particular goal of an adversary. When an attacker is interested in a specific organisation because of its footprint (e.g. the financial sector due to its negative reputation), the strength of the balancing loop B6 Demotivating may be completely offset, giving way to escalatory attack patterns (II.C.2).

Table 4: Summary of Results in 4.3 Adversary Dynamics

Findings	Empirical Evidence	Relevance
Code review as one of the most effective mechanisms should be done in pairs but the learning experience is dropped due to time constraints.	Q 24; II.C.3	Along the same line of Rahmandad, & Repenning learning from mistakes and code review have been pointed out for improving security. Yet, time constraints prohibit this activity.

Since vulnerabilities may be exploited fast, stepping back, taking time, and reframing the situation to address the issue may be counterproductive.	GMB;
	Q 25, 26;
	II. C. 2, 3; II. D. 15
		Similar to the findings from Rudolph, & Repenning (2002), fast actions and a reduced mean time to resolve are necessary to decrease the number of exploitable vulnerabilities.

The probability of a successful attack depends on the state, quantity, and criticality of vulnerabilities. Lower criticalities are often not fixed and also the larger quantities of those are often not considered as dangerous.	GMB;
	Q 27, 28;
	II. D. 4, 6, 15
		Next to fixing critical vulnerabilities, organisations need to also fix vulnerabilities of lower criticalities instead of merely adding those to the backlog. Since quantity-focused attacks may be successful, fixing is necessary.

Cyber adversaries are like private organisations, employ similar methods, and follow a business case.	GMB; Q 29, 30	Understanding cyber adversaries as business organisations may help to reframe knowledge in science and practice when attempting to understand attackers.

Cyber adversaries build capabilities in an escalatory pattern. Only demotivating leads to counter-acting this spiral.	GMB;
	Q 31, 32;
	II. C. 2, 3
		Since private organisations have only successful defence as demotivating strategy, governments need to step in to chase and arrest attackers.

4.4 Organisational Attack Response

Since attacks may be understood as development projects, they require time to process and pass all the necessary steps. Additionally, when attackers attempt to exploit a vulnerability through malware or hacking, they do not only need to find this vulnerability and know how to abuse it but they also have to pass an array of preventive defence mechanisms and have to make sure that they stay undetected because otherwise the attack would turn unsuccessful. Common prevention and detection mechanisms include firewalls, blacklists, proxy servers, antimalware software, anomaly scanners or security event monitoring (e.g., Ahmad et al., 2014). If an attack is stopped through prevention mechanisms, no further action from the side of an organisation is required. However, if detection mechanisms show the intrusion of an external malicious actor into an organisation's ICT system, manual actions from the organisation's Computer Emergency Response Teams (CERT), further experts, and the DevOps team which is responsible for the attacked area of the organisation's ICT system are necessary, either to halt the attack, or to pick up the pieces, learn from the incident, and improve the organisation's future defence. The number of *DevOps needed for Attack Response* depends on the specific case (Figure 12). Generally speaking, a large number of people is involved at the very beginning to find the root cause of the problem. Once the underlying reasons are found, a smaller team of experts works to fend off the attack. In both phases of the response, mainly the most

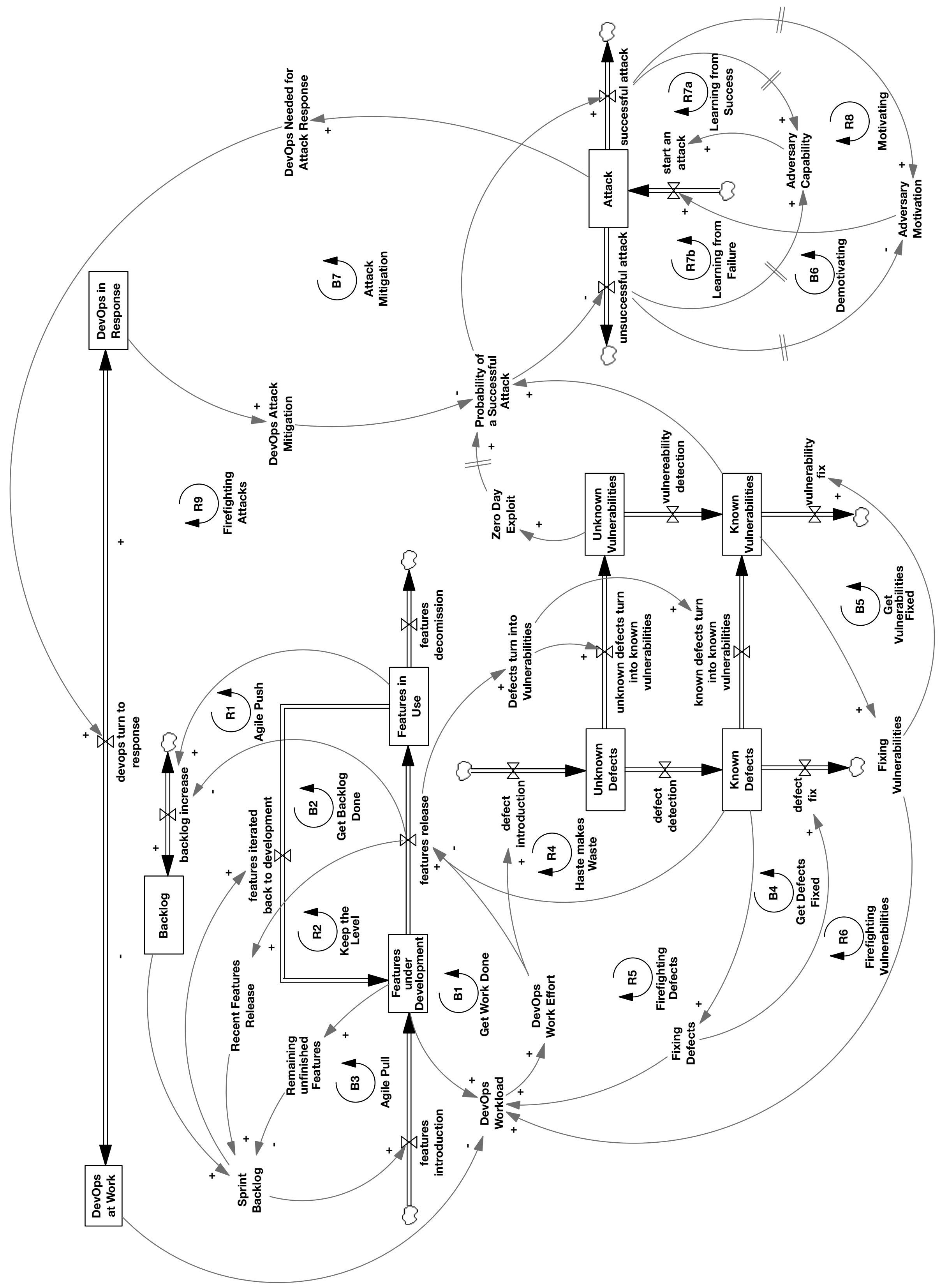


Figure 12: Causal Structure of Organisational Attack Response, added to the causal structure from Figure 11 (Based on the broader literature incl. Rahmandad, & Repenning, 2016; Repenning, 2001; Repenning, & Sterman, 2002; Sterman, 2000, Martinez-Moyano et al., 2015 and secure software development literature, and empirical findings within the organisation). While previous research on software development (or broader product development) and defects illustrated similar structure, the connection from defects to vulnerabilities, the connection from vulnerabilities to adversarial dynamics, the way how to depict adversarial dynamics, and the connection between work and reaction has not been investigated previously.

experienced DevOps from the responsible team and further experts are occupied with the tasks of mitigating the attack. No matter the overall number of DevOps needed for

Attack Response, the desired number of *devops turns to response*. Increasing numbers in the stock of *DevOps in Response* improve the attack mitigation capabilities of an organisation, and thereby, decrease the Probability of a Successful Attack. Hence, *DevOps Attack Mitigation* leads to less successful and more unsuccessful attacks, thereby creating the balancing feedback loop *B7 Attack Mitigation*. While the mechanism of this balancing feedback loop appears to be a simple and reliable way to counteract the escalatory attacker behaviour, the participants during the workshop emphasised its limitations:

> *33. When you respond to an attacker who is already in, often you are too late. What you do is you try to reduce the impact, learn from it, and go on.* (II.C.3 116 ff.)

In other words, responding to an attack is the last resort, and while trained and prepared with CERT, DevOps, and further experts, on its own it does neither represent a reliable, nor a desirable mechanism to address the security risk from cyber attacks. Additionally, since devops turn to response in case of a detected cyber attack, fewer DevOps are available to conduct the regular work of developing and operating software, thereby further increasing the DevOps Workload in a situation which is already governed by pressure. In an effort to keep up with the work, the teams strengthen their work effort. While this may reduce the pressure arising from the Features under Development, it also leads to more defects which later turn into vulnerabilities, both causing more, and potentially unknown and not recognised work in the future. Similar to the firefighting mechanism explained earlier, the lack of attention to upstream activities due to high pressure, such as careful design, abuse cases or code review in pairs, causes the DevOps to firefight at a last stronghold against the attempts of an external adversary to maliciously exploit the previously introduced vulnerabilities. Since an attack is disproportionately costly (Anderson et al., 2013), organisations attempt to avoid successful breaches by all means. Hence, while it is common that organisations respond to an attack, the interactions between pressure, software development, software vulnerabilities, and cyber attacks make it likely that teams and organisations become trapped in a persistent mode of the reinforcing feedback loop *R9 Firefighting Attacks*. The pressure arising from this downward spiral is reinforced by the lack of senior DevOps within the teams at work which decreases the productivity of the regular development and operation tasks (not indicated in the diagram for simplicity reasons). During a meeting with the collaborating security team, one of the experts referred to an other organisation which recently had a similar, though planned case:

> *34. They had more than a thousand people in the warroom for an entire day. This was a planned exercise for training but the business impact was still heavy. I can really imagine that this has a strong effect on companies.* (II.D.17 182ff.)

In the same and further meetings, it has been pointed out that this mechanism is real and dangerous but it has also been emphasised that it represents a rather extreme case because larger organisations have enough resources to buffer such attacks (e.g., II.D.23). Several hints from the empirical insights and previous findings described in literature modulate the probability of this mechanism: First, DevOps teams normally have a standby agreement with other teams to cover each other in case of emergencies, in both business and security. Some DevOps have indicated problems in this context:

> *35. They [another team] don't do standby, so if there is an issue we have to work quite hard. I think, we should escalate that to a higher level.* (II.D.11 71ff.)

Hence, in case of an emergency, problems may arise from time delays in response, pressure, or understaffing. Second, response activities may last for a day or two, but it may also take several months in which the defender and attacker interact in a cat-and-mouse game. One of the collaborating DevOps teams, for instance, described a business incident (i.e., the DevOps were in response mode due to functionality problems and not security issues) which lasted for several months and clearly emphasised that there is no desire to come back to such a situation (II.D.11). Particularly in the case of long-lasting response activities, the productivity of DevOps teams is severely affected and pressure rises even further, potentially causing more defects and vulnerabilities which reinforce the vicious circle. Third, smaller organisations are unlikely to have the necessary resources to conduct a response lasting several months. An ethical hacker within the organisation described another company which was not able to fend off the attacker alone:

> *36. Battles between hackers and defenders can go over several months. Often you think the guy is out but then he was just stealthy or had a backdoor or something else and is still inside. […] there was this case of a smaller company where the people went back to work and the attacker was still inside because they could not afford it anymore to neglect their normal jobs* (II.D.2 173ff.)

In the described case the business impact of not developing features anymore had become more harming than the cyber attack itself. Eventually, the firm had solved the problem with the expensive help of an external security provider who was hired as a last resort (II.D.2). Fourth, a security expert speculated about the danger of targeted attacks. While cyber criminals may have no interest of deliberately pushing an organisation into a persistent mode of firefighting, state actors could aim to destabilise a country or region by collapsing a private organisation in the course of cyber warfare (II.D.22). Fifth, several heuristics, biases and misperceptions obscure causes and consequences of decisions and actions. It was previously emphasised that employees may underestimate the real danger of dynamic mechanisms in cyber security due to well known availability heuristics (Tversky, & Kahneman, 1974). Similarly, Repenning and Sterman (2002) found out that managers are subject to attribution errors, meaning that they ascribe the cause of problems in production to employees' inadequate work effort, and not to the dynamics of pressure and firefighting. Interestingly, during the six months on site, several experts from the security field and managers have complained about lacking compliance by the DevOps. So, while DevOps may underestimate the actual problems of software complexity, shortcuts, defects, and vulnerabilities, managers may not recognise that declines in productivity occur (at least partially) due to overwhelming pressure and firefighting activities and not because of a lack of compliance from the side of the DevOps. Along the same line, managers, security experts, and DevOps may state that an organisation applies agile software development, whereas in practice, the organisation engages in dangerous cherry picking. As pointed out by MacCormack and colleagues, "to the degree that such a process relies on a coherent system of practices, a piecemeal approach is likely to lead to disappointment" (MacCormack et al., 2003, p. 84). Confronted with this idea, the responsible expert for secure software development admitted the problem:

> *37. Among my colleagues throughout many industries, this is a big question. Nobody knows whether we actually apply agile or whether we simply call it like this and do instead something else.* (II.D.23 207ff.)

Combined, availability heuristics, attribution errors for causes of problems, and unconscious differences between what organisations state they do (i.e., conducting agile software development) and what they actually do (i.e., taking some of the mechanisms of agile methods and combine them with other approaches)[30] may increase the likelihood of pressure and firefighting. Last but not least and as explained above, Rahmandad and Repenning (2002) demonstrated the potential of capability erosion when attempting to optimise resource allocation in an uncertain environment. While the internal environment of the software vendor in their study was already governed by uncertainty about the future, actions from a malicious cyber adversary are likely to be even more uncertain.

[30] This is framed as espoused theory and theory in use (see e.g., Vennix, 1996 referring to Argyris, 1992; and Argyris, & Schön, 1978).

In short, empirical evidence indicated that the occurrence of persistent firefighting regarding vulnerabilities and attacks is not an inescapable route. Yet, the findings also suggested that dismissing the potential of firefighting vulnerabilities and incidents entirely is a dangerous and negligent fallacy. Like organisational accidents, cyber attacks build up by many small pieces that eventually cause the attack to be successful (Goh et al., 2014; Lacey, 2009). Governed by several counterproductive efforts of firefighting to solve pressure and reinforced by the escalatory behaviour of adversaries, organisations may be more vulnerable than they first appear. Having built on Rahmandad and Repenning (2016), the true capacity of developing and operating software within an organisation may even be lower than proposed by the authors in their study, as shown in Figure 13. In other words, this study points out that an organisation subject to a dual firefighting mechanism and escalatory behaviour may have two apparent and one true system which blurs the actual capacity. Hence, the interaction of pressure in software development, software vulnerabilities, cyber attacks, and response has the potential to severely affect the trade-off between software functionality and software security: While security is sacrificed for functionality in an effort to address the short-term business risks, the overall risk in both security and subsequently business rises due to the lacking focus on security. After an initial gain from the focus on functionality, the growing number of vulnerabilities and subsequent attacks potentially force the DevOps into persistently firefighting vulnerabilities and probably even attacks which causes the DevOps to drop their regular work, eventually disrupting business and harming the organisation's performance. In the words of one of the security experts in software development:

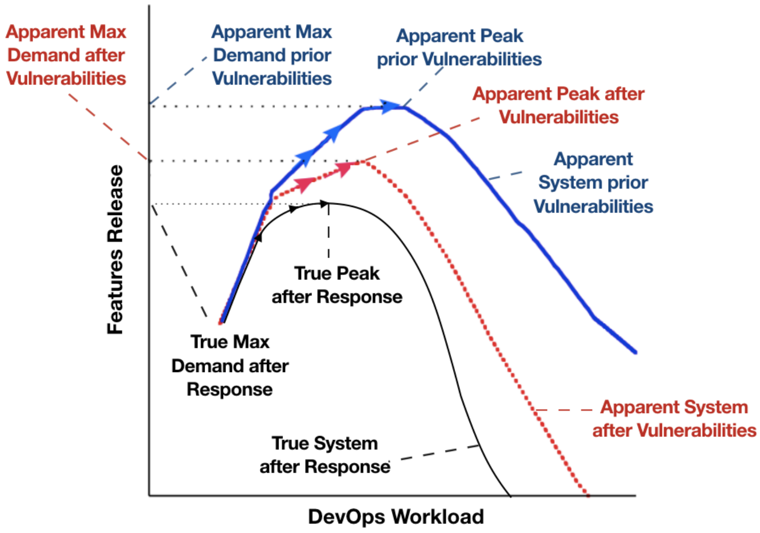


Figure 13: Suggested Behaviour in a System with two apparent Performance Optima. The system is governed by the underestimation of potential dangers, several mechanisms of misperception, and the interaction between two firefighting spirals and escalatory patterns (based on Rahmandad, & Repenning, 2016 which only had one apparent and one true system). The blue and red line describe the apparent, the black line the real relationship between pressure (DevOps Workload) and performance (Features Release).

> *38. If stress is high, quality and particularly security go down which costs a lot in the long run. In fact, cheap is always expensive in the long term. Eventually, you need security anyways so do it right from the beginning! When you do it later, it costs more, it is harder, and it harms your business.* (II.D.15 146ff.)

In the light of the array of reinforcing mechanisms inside and outside an organisation, pressure is likely as the possibilities for stress are abundant. Hence, it is necessary to take a conscious and informed decision when trading off software functionality and software security and manage the arising risk. An external executive underlined the importance of matching an organisation's internal approach with the external environment to be successful in business:

> *39. Business is about risk. If there is no risk, there is no business because everybody would simply do it. Cyber attacks are just another kind of risk we have to deal with.* (II.D.13 8ff.)

Table 5: Summary of Results in 4.4 Organisational Attack Response

Findings	Empirical Evidence	Relevance
Responding to attacks is necessary but may catch DevOps in a persistent firefighting mechanism, particularly if pressure is already high, the attacker is powerful, and large amounts of unexpected work arise.	Q 33, 34	Despite the danger from firefighting mechanisms, it is vital to respond to attacks because of the disproportionate costs arising from successful cyber attacks. Meanwhile, production pressure must be reduced.
Combining the empirical hints from several areas (deviations in security standards, potential time horizons of response, the size of an organisation, targeted attacks, availability heuristics, attribution errors, espoused theory and theory in use, and adaptation traps) sheds new light on the probability of firefighting vulnerabilities and attacks.	Q 35, 36, 37 II.D.2, 19, 22; Rahmandad, & Repenning, 2016; Repenning & Sterman, 2002	While described as possible but improbable, literature and empirical findings suggest that firefighting vulnerabilities and even attacks may be more likely than initially assumed. Under these conditions, capability erosion becomes more likely as there are two apparent systems that hide the real one.
Trading off security for functionality may feed back later and turn out to increase both, business and security risks, and swallow earlier profits. Thus a conscious decision about the true risks is necessary.	Q 38, 39 McGraw, 2006, 2012	Along the same line as previous findings in literature, "building security in" is cheaper in the long-term. The firefighting and escalatory dynamics may lead to business disruptions decreasing performance.

5. DISCUSSION AND CONCLUSION

In the last decades public and private organisations have embraced ICT and particularly software to improve their performance and achieve success (e.g., Wade, & Holland, 2004). Nevertheless, software depicts a critical entry point for malicious cyber adversaries external to an organisation because of the abundant, and continuously rising number of global software vulnerabilities which enable successful cyber attacks (Ablon, & Bogart, 2017; Verizon, 2016). Software vulnerabilities are caused by defects in the coding or configuration of a software. Such defects arise inter alia from taking shortcuts due to pressure when developing and operating software (McGraw, 2006, 2012; Oliva, & Sterman, 2001; Rahmandad, & Repenning, 2016). While agile approaches in software development are theoretically not supposed to allow pressure, also DevOps teams following agile methods are subject to limited resources. This forces them to prioritise activities and consequently trade off software functionality to address short-term business risks from market pressure against software security to cope with long-term security risks from cyber attacks. Considering research from the organisational theory and strategy literature (e.g., Levinthal, & March, 1993), such trade-offs between the short-term and the long-term create pressure. Despite this contradiction to theory in agile methods, there have been no studies which connected pressure in agile software development, software vulnerabilities, cyber attacks, and organisational response to better understand and explain the trade-off between software functionality and software security. Hence, this research set out to answer the following research question by conducting a model-based case study in collaboration with a European financial organisation:

> *How does the interaction between pressure in software development, software vulnerabilities, external cyber attacks against an organisation, and the organisation's attempt to mitigate those attacks influence the trade-off between software functionality and software security?*

Overall, this study provides seven contributions and explains several theoretical and practical implications for software security and cyber attacks. The first contribution of this study is a rich description and explanation of the causes and consequences of the dynamic interplay between pressure in agile software development, software vulnerabilities, cyber attacks, and attack mitigation which is laid out below. While the description of this interaction represents a contribution in itself as it has not been done before, there are four specific points which are clearly named and emphasised in the text as individuals contributions.

Despite the self-organising nature of DevOps teams (Schwaber, 2004), the findings from this study show that pressure also exists in agile software development. To address their workload, DevOps particularly increase their work effort. While this enables teams to release more features and achieve fast value as aimed for in agile methods, increased work effort also causes more defects. In contrast to previous findings (Repenning, 2001), fixing these mistakes does not describe a firefighting mechanism as in product development but is part of the nature of agile software development due to their short cycles. Consistent with criticism raised in literature (e.g., Becker, 2014; Heitzenrater et al., 2016; McGraw, 2006, 2012; Neumann, 2012), it is, however, commonplace that long-term benefits from security are sacrificed for short-term gains from functionality when fixing errors, meaning that particularly functionality-related issues are addressed while security defects are neglected, leading to vulnerabilities upon release. Hence, increased pressure causes more defects which in turn leads to higher numbers of vulnerabilities. In this sense, the second contribution of this study is to explicitly draw the connection between pressure, defects, and vulnerabilities in a systematic way. The responsible for secure software in the financial organisation noted that he would not believe that more pressure causes more vulnerabilities, but illustrating the connection from pressure to defects to vulnerabilities is a simple, yet clear way of laying out this pathway to weakness.

In contrast to fixing defects, solving vulnerabilities has the potential to intensely disrupt the regular activities because software vulnerabilities have to be fixed fast as they may be exploited by external attackers. While it is unlikely that a vulnerability is exploited immediately, the need to solve weaknesses in a timely manner still reinforces the pressure resting on the DevOps teams, potentially causing them to blunder into a mode of persistent firefighting or even organisational collapse. Thus, solving the problem is urgent but once the issue is addressed, the solution also creates new problems. In this sense, this study confirms previous findings that the usual approach to cope with complex problems (i.e., taking time, learning about the issue, and addressing it later) has the strong potential to exacerbate the issue instead of improving it when used in an environment of immediate danger (Repenning 2003; Rudolph, & Repenning, 2002). At the same time, such necessary, but continuous firefighting may lead to the permanent erosion of software development and operation capabilities due to the wrong adaptation to future workloads (Rahmandad, & Repenning, 2016). In short, the third contribution of this study is to explicitly link software vulnerabilities and the probability of a successful cyber attack since this connection poses a dilemma to an organisation: It has to fix the vulnerabilities fast as they may be exploited by an external adversary but by fixing them rapidly, the organisation may get caught in persistent firefighting. Interestingly, previous research has either looked at firefighting mechanisms (e.g., Rahmandad, & Repenning, 2016; Repenning, 2001), or fast problem solving (e.g., Rudolph, & Repenning, 2002) but did not link those two elements. Such a combination is reasonable though as it has been shown in this study that issues may be simultaneously subject to severe time pressure and pressure from workload.

In contrast to previous research on the dynamics of firefighting, this study included an external actor who aims to exploit potential weaknesses of an organisation. To this end, cyber adversaries follow an objective, have a business case, acquire resources, and cultivate capabilities like any private organisation as well. In this sense, they simply join the ranks of competitors an organisation needs to address, with the small difference that these new competitors play by different, illegal and unpredictable rules (Figure 14). Consequently, organisations have to broaden their perspective on competition and integrate business and security strategies to address the threats from malicious cyber adversaries.

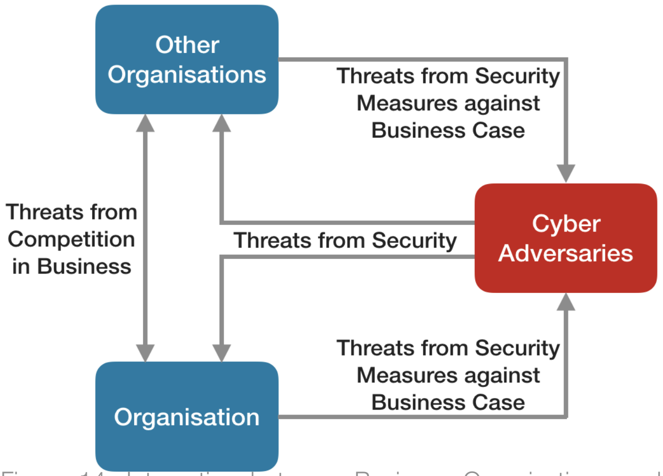

Other Organisations
Threats from Security Measures against Business Case
Threats from Competition in Business
Threats from Security
Cyber Adversaries
Threats from Security Measures against Business Case
Organisation

Figure 14: Interaction between Business Organisations and Cyber Adversaries. Business organisations threaten each other through economic competition and are additionally threatened from cyber adversaries. Organisational defence mechanisms threaten the business case of cyber adversaries.

Additionally, cyber adversaries are following several escalatory patterns, similar to those observed in terrorism research (Martinez-Moyano et al., 2015), which reinforce their activities, enhance their capabilities, and increase their motivation, likely leading to more future attacks. Understanding the interaction between cyber adversaries driven by escalatory behaviour and an organisation as a form of competition that needs to be addressed by integrating business, ICT, and cyber security strategies depicts the fourth contribution of this study. Research on competition in the fields of strategy and organisational theory has applied terms commonly used in cyber security which provides a further hint of the benefit of incorporating the different fields in an integrated idea of strategy. In this sense, Porter for instance wrote that “awareness to these forces can help a company stake out a position in its industry that is less vulnerable to attack” (Porter, 1979).

If not stopped by preventive measures, organisations have to respond to those attacks to avert successful breaches which would lead to disproportionate costs (Anderson et al., 2013). Even if an attack is successfully warded off, additional pressure arises because DevOps, and particularly the seniors, interrupt the pursuit of their regular tasks, potentially fuelling the downward spiral of firefighting software vulnerabilities and cyber attacks. The findings of this study indicate that such dual firefighting mechanisms against both internal and external problems exist and are more likely than expected. Next to organisational or attack characteristics, particularly cognitive limitations and misperceptions lead to problems. Paraphrasing Porter and Millar (1985, p. 152), technological solutions have changed organisational and environmental opportunities and threats faster than employees and managers can explore, analyse, and react on. Thus, misperceptions arising from availability heuristics, attribution errors, and the difference between espoused theories and theories in use increase the likelihood of the occurrence of a vicious circle because people often only realise the existence of problems and firefighting mechanisms once it is too late (Repenning, 2001; Repenning, & Sterman, 2002; Tversky, & Kahneman, 1974, 1986; Vennix, 1996). Along the same line, the system’s physical and decision structure further complicates the detection and confrontation of firefighting since it is characterised by two apparent performance optima (i.e., prior to the detection of vulnerabilities and prior to attacks) which both represent a pathway to vicious circles as an organisation may adapt to underestimated future workloads. Hence, adding an external actor to the system potentially increases the effects of previously discovered traps in production and development (Rahmandad, & Repenning, 2016; Repenning, 2001; Repenning, & Sterman, 2002; Rudolph, & Repenning, 2002). Thus, it is the fifth contribution of this study to include an external adversary which leads to the potential of a dual firefighting mechanism and two apparent performance optima which blur the real capacity in the system. These findings strengthen the insights from previous research in firefighting and capability erosion (Rahmandad, & Repenning, 2016; Repenning, 2001).

In short, the interaction of pressure in software development, software vulnerabilities, cyber attacks, and organisational response plays out as follows: Pressure appears to exist in agile software development. While it increases performance, it also causes defects, leading to vulnerabilities which allow an external adversary to exploit an organisation. Continuously fixing defects and vulnerabilities, and responding to attacks potentially leads an organisation into a permanent downward spiral, causing the DevOps to drop their regular work, particularly in the light of escalatory attack patterns described by adversaries. The sixth contribution of this study is to connect the previously described interaction with the trade-off between software functionality and software security, thereby answering the initial research question. Favouring software functionality to address business risk over software security to address the risk from cyber adversaries may only lead to a temporal success. Initial gains from functionality may be offset by rising security problems that disrupt the development and operation of software and thereby harm the organisation's performance. Stated differently, the interaction between pressure in software development, software vulnerabilities, cyber attacks from an external adversary, and organisational attempts of attack mitigation create a dynamic complex system in which the short-term outcome is likely to be very different from the long-term state. Favouring software functionality enables an organisation to address threats from competition in business. Meanwhile, this decision makes an organisation vulnerable to cyber attacks from an external adversary. Even when not accounting for the enormous costs of successful cyber attacks, the pressure from addressing security threats by fixing software vulnerabilities and mitigating cyber attacks may trap the organisation in a persistent mode of firefighting. As a consequence, less focus on functionality is possible as more and more DevOps are concerned with security-related problems. Thus, the organisation shifts its focus from addressing threats from business competition to threats from security. Hence, initial short-term gains may very well be lost by the long-term consequences arising from the dynamic complexity of the system. While neither the details of the trade-off, nor the strength of the previously described interplay are considered here as no quantitative analysis has been conducted (Homer, & Oliva, 2001; Sterman, 2000), in the light of the explained interaction it is clear that neglecting software security for short-term gains from software functionality represents a better-before-worse scenario often addressed in the field of system dynamics (Braun, 2002; Senge, 1990; Sterman, 2000). In sum, the interaction between pressure in software development, software vulnerabilities, cyber attacks from external malicious adversaries, and organisational attack response affect the trade-off between software functionality and software security in such a way that short-term benefits may turn into permanent long-term losses as an organisation may get trapped in a dual firefighting mechanism while the adversary escalates his/her attack patterns to benefit from the weakened organisation.

5.1 Theoretical and Practical Implications in Software Security

Since this research was conducted by combining practice-driven scientific approaches and in collaboration with a European financial organisation which actively uses the results of the research, this study provides both theoretical and practical implications for scientists and practitioners in the field of agile software development and cyber security. The implications are presented in four steps, discussing pressure, defects and vulnerabilities, the trade-off between software functionality and software security, and adversarial dynamics.

5.1.1 Implications regarding Pressure

In contrast to theory (Beck et al., 2001; Pressman, 2010; Schwaber, 2004), agile software development does not appear to be without pressure. Hence, productivity and outcomes may be worse or at least different than expected as the success of development practices depends on the consistent application of those (MacCormack et al., 2003). This study urges to scrutinise whether organisations truly apply agile methods, or whether they simply state their intents and simultaneously act in dangerous cherry picking (MacCormack et al., 2003). Indications for the latter case are backroom politicking, exhausted staff, the separation of development and operations, DevOps receiving managerial directives that inhibit the self-organising nature of the team, or unfinished products at the end of a sprint (Schwaber, 2004). Since people or organisations generally do not recognise the difference in their stated and their actual behaviour (Vennix, 1996), it may be challenging to investigate the extent to which the espoused theory deviates from the theory in use. Moreover, it could even be difficult to recognise pressure in the first place since habits or corporate culture could blur the inquiry. In this context, the comment from one of the DevOps who had just recently joined the respective team is illuminating because he pointed out the theoretical benefits but practical lack of pairwise code reviews, a fact that others of the team seemed to had forgotten about (Quote 24; II.D.11). Additionally, the common practice in agile approaches to delay tasks may further obscure the true extent of pressure within an organisation since the work which creates pressure is simply put on hold. In practice, a growing backlog across all teams could indicate such a scenario. Next to blurring the true picture, findings from this research confirm the previously described danger of constantly postponing tasks (van Oorschot et al., 2013), because this practice eventually results in a costly strategic delay and future schedule pressure. While previous research has indicated that such time pressure cannot be solved by increasing the number of DevOps (Arora et al., 2006, referring to Brooks, 1995), this study described the practice in agile software development of deliberately dropping less important projects to "cut the pressure loop" by splitting tasks and teams and adding new DevOps to the project (II.D.23 148). Since it is realistic to assume that the overall pressure has not been completely solved once it comes to downstream activities in software development (Rahmandad, & Repenning, 2016), this measure of "cutting the pressure loop" is highly relevant for practice in software development as it fulfils the need to address problems fast and yet avoids firefighting mechanisms.

5.1.2 Implications regarding Defects and Vulnerabilities

While "there are always some bugs in the released software" (Rahmandad, & Repenning, 2016, p. 654), in recent years, automated testing has increased the effectiveness and efficiency of tests in software development, leaving fewer unknown defects behind. The findings of this study indicate though that investments in automated testing are void if detected errors are not fixed afterwards. While this sounds obvious, the trend in the financial industry to layoff employees due to technical possibilities (Crowe, 2016; Lopez, 2013; Rankin, 2013) may impede the actual impact of automated testing. At the same time, it is noteworthy that automated testing leads to two further benefits: First, even if defects are not fixed after having been detected, automated testing enables an organisation to have a better understanding of the actual future amount of work to avoid wrong adaptation, thereby potentially preventing firefighting mechanisms. Second, an external expert indicated the opportunity of employing automated testing in real time while developing software, allowing DevOps to receive fast feedback while coding (II.D.14). Particularly the second benefit would enable DevOps to decrease the number of defects prior to release and subsequently reduce the amount of software vulnerabilities. In line with previous research (Ablon, & Bogart, 2017; CVE, 2017; McAfee, 2014; Verizon, 2016), this study points out the growing number of software vulnerabilities and emphasises the danger of only focusing on vulnerabilities with a critical or high severity. In practice “the numbers count” (II.D.15 42), and thus even leaving vulnerabilities with a lower criticality open may jeopardise an organisation’s success. In this context, experts within the financial organisation have underscored the relevance of decreasing the mean time to resolve. This request received particular weight through the two previously mentioned cases of WannaCry and NotPetya that exploited a well known software vulnerability in Microsoft’s operation systems. Although the software vendor had provided a security patch long before, many organisations worldwide were still vulnerable at the time of the attacks (Fox-Brewster, 2017).

5.1.3 Implications regarding the Trade-Off between Functionality and Security

Throughout the last decades, Gary McGraw has relentlessly emphasised the need to “build security in” (2006, 2012). Although further research showed that addressing defects during the upstream phases of software development is much easier and cheaper (Boehm, 1994; Stecklein et al., 2004), practice has frequently not followed this request, supposedly because of economic reasons (Bojanc, & Jerman-Blažič, 2008). Although this study has neither conducted computer simulations of scenarios, nor financial analysis of software development, the answer to the research question indicates that sacrificing software security for short-term gains through software functionality eventually causes lower financial returns due to the feedback effects arising from the dynamic complexity of the system. Building on the theory of the adaptation trap (Rahmandad, & Repenning, 2016), this study showed that the dual firefighting mechanism to cope with both software vulnerabilities and cyber attacks increases the likelihood of trapping an organisation in a downward spiral, potentially causing permanent capability erosion. Finally, since capabilities are learned and continuously practiced activities that take time to accumulate (Dierickx, & Cool, 1984; Rahmandad, 2012; Winter, 2003), it may be questionable whether organisations that only focus on software functionality even have the capability to include software security, fix software vulnerabilities, and respond to cyber attacks. Considering previous research, it appears that “application developers usually do not have expertise in security“ (Hamid, Gürgens, & Fuchs, 2015, p. 109), thus, software security capabilities are scarce in organisations (McGraw, Migues, & West, 2016). Consequently, this study renews the call to balance software functionality and software security from early on and "build security in". Insights from this study show that a first starting point to pursue this aim is to increase the security awareness among managers, and to improve the maturity of DevOps. Consistent with previous literature (Repenning, & Sterman, 2002), experts within the financial organisation explained that such efforts of creating awareness and improving processes would first decrease productivity due to the time spent in trainings to later improve the overall performance (II.C.3). It is noteworthy that due to the misperceptions described earlier, trainings may only lead to success if both managers and employees are considered.

5.1.4 Implications regarding Adversarial Dynamics

In line with previous research (Libicki, Ablon, & Webb, 2015), this study understands cyber adversaries as strategically thinking actors. Moreover, this study proposes to recognise cyber adversaries as another kind of competitor an organisation needs to be aware of. While this recognition is not meant in a moral sense, on a strategic level an organisation needs to integrate business, ICT and cyber security strategies to address the full range of threats from market and security competition. As ICT does not created value without being part of the overall business strategy of an organisation (Bharadwaj, 2000; Brynjolfsson, & Hitt, 2000; Henderson, & Venkatraman, 1993; Johnston, & Carrico, 1988; Kettinger et al., 1994), security has little effect if it is not integrated into the overall strategy. To this end, understanding security risks as equally important as business risks represents a first step forward. Since it is difficult though to develop strategies that include the uncertain actions of cyber adversaries and the interaction between attackers and defenders, models integrating the dynamic complexity of the interplay inside and outside of an organisation may help to inform decision makers (Cosenz, & Noto, 2016; Gary, Kunc, Morecroft, & Rockart, 2008; Porter, 1993).

5.2 A Theory on Vulnerability Dynamics

The findings described above indicate that trade-offs between different objectives (e.g., software functionality and software security) cause pressure, leading to errors and subsequent vulnerabilities which may be exploited by an external actor. While organisations may decide to stop regular processes and react on the incident to avoid an exploitation, fixing vulnerabilities and mitigating attacks may trap them in a dual firefighting mechanism. These findings do not only apply in the case of software development and cyber security, but can also be generalised for a broader range of situations. Hence, based on the insights gained within this research and on the broader literature from several different fields, in a last step, this study applies Occam's razor and generalises its findings to move from case specific statements in the area of software development and cyber security to generally applicable explanations about the dynamics of pressure, vulnerabilities, firefighting, and escalation. Thus, the study provides necessary conditions and testable propositions for an explicit theory on vulnerability dynamics, thereby further increasing the external validity of the study (Forrester, 1961; Kopainsky, & Luna-Reyes, 2008; Pidd, 2003; Yin, 2014).

In total, there are seven necessary conditions for the occurrence of vulnerability dynamics: First, limited resources force an organisation to trade-off between at least two different gaols. As described above, it is likely that goals for improving short-term performance are favoured over long-term issues. Second, similar to Rahmandad and Repenning (2016), resources are used to address all of the previously traded off goals (a concept called resource fungibility). The third and potentially most obvious conditions describes that if errors do not make an organisation vulnerable, no vulnerability dynamics occur. Fourth, an organisation needs to have the possibility to address the vulnerabilities prior to potential attempts of exploitation because this opportunity causes the previously described dilemma between preventing successful attacks and firefighting. Fifth, from the perspective of the organisation, there must be an external actor who is willing to take advantage of any kind of weakness the organisation is subject to. Sixth, due to changes in motivation and learning from the outcomes of activities, an external actor has the potential to follow an escalatory behaviour of attempted and probably successful actions. Finally, the organisation must be able to trade off between continuing with its regular activities and reacting on attacks because this opportunity causes the dual firefighting mechanism described above. Table 6 summarises the necessary conditions for vulnerability dynamics to occur.

Table 6: Summary of Conditions for Vulnerability Dynamics

1	Resource constraints force to trade off between at least two different objectives.
2	Resource fungibility.
3	Errors make vulnerable.
4	Possibility of addressing the vulnerability before somebody can take advantage of it.
5	(Malicious) external actor who could prey the weakness.
6	Potential for escalatory behaviour from the external actor due to changes in motivation and capabilities.
7	Trade-off between continuing with regular activities or reacting on the actions of the external actor.

To make the generalisability of the research's findings explicit, the study presents a simplified and generic causal diagram (Figure 15) about the dynamics of pressure, vulnerabilities, firefighting, and escalation applied to the case of the French urban riots from 2005 (Chrisafis, 2015) which was chosen as all of the previously explained conditions are matched.

In 2005, France was subject to the most extreme demonstrations and riots in its entire contemporary history. According to Mucchielli, “the scenario has been more or less the same since the first ‘urban riots’ in 1990 and 1991 […] The riots were triggered by the death (intentional or accidental) of local youths connected (in various ways) with police intervention” (Mucchielli, 2009, p. 734). In the case of the French riots from 2005, police officers were chasing a few youths in the neighbourhood of Clichy-sous-Bois, a worker-class and so called ‘problem urban area’. Coincidently, three other adolescents appeared on the scene but became afraid of the police officers and fled into a power transformer. The three youths were seen “by at least one police officer whose superiors apparently felt that he and his colleagues had more important things to do than to attend to the boys, although their lives were in danger” (Mucchielli, 2009, p. 736). As a consequence of not helping the three youths but instead continuing the initial chase (B1), two of the three youths died in the power transformer which caused outrage and riots in the area (R1). Attempting to solve the problem and decrease the public pressure, the French administration denied any responsibility for the fatal accident and even accused the adolescents of having committed a crime which caused their deaths (B2), unintentionally leading to more riots which started to spread over France. Trying to control the situation, police forces turned against the protesters (B3). Since all of the efforts to keep the situation under control reinforced pressure (R2, R3), tension increased and eventually resulted in a tear-gas grenade being thrown in front of a mosque in Clichy-sous-Bois. Thereafter, the simmering conflict between people from ‘problem urban areas’ and the French state exploded and led to escalating riots in the entire country (R4) and extreme police interventions (B4). The tension grew so high that the government used every mean to maintain public order (R5), for instance, when the Interior Minister Nicolas Sarkozy called for a ‘Kärcher’ to clean neighbourhoods of ‘scum’ (Sciolino, 2007).

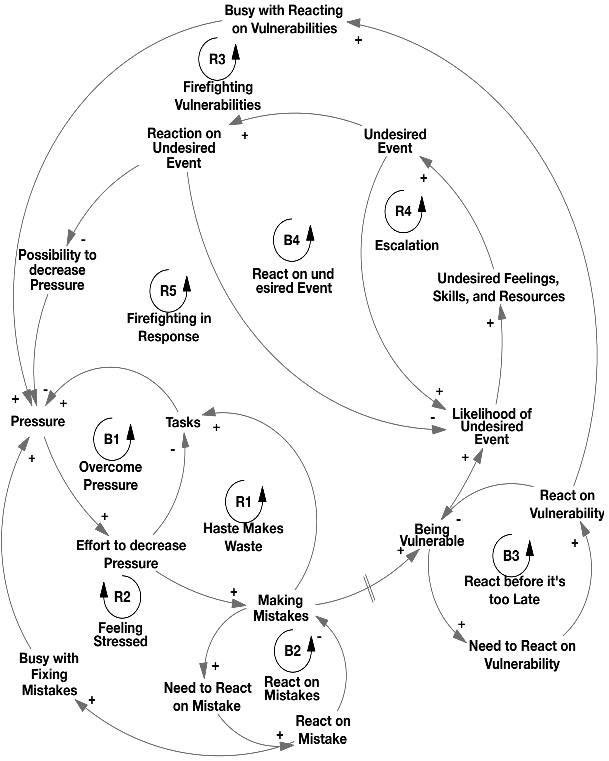

Busy with Reacting on Vulnerabilities
+
R3
Firefighting Vulnerabilities
Reaction on Undesired Event
+
Undesired Event
+
R4
Escalation
B4
React on und esired Event
-
Possibility to decrease Pressure
R5
Firefighting in Response
Undesired Feelings, Skills, and Resources
+
+
-
Likelihood of Undesired Event
+ - +
Pressure
B1
Overcome Pressure
Tasks
+
-
+
+
R1
Haste Makes Waste
React on Vulnerability
-
Being Vulnerable
+
+
B3
React before it's too Late
+
Effort to decrease Pressure
R2
Feeling Stressed
+
Making Mistakes
+
Need to React on Vulnerability
B2
-
React on Mistakes
Busy with Fixing Mistakes
+
+
Need to React on Mistake
+
React on Mistake

Figure 15: Causal Diagram of a Theory on Vulnerability Dynamics (Derived from the Generalisation of the Study’s findings).

Despite obvious differences, such as that the riots eventually abated, the overall course of the events in France was very similar to the case described above for secure software development and cyber attacks: Under pressure, police officers took a shortcut and did not help the three youths. This mistake caused further pressure. Although the government engaged in various firefighting activities, such as large police operations, the situation eventually escalated and the country fell into chaos, further strengthening the downward spiral. While the roots of the demonstrations are likely to be based in an economic, social, political and identity crisis of large parts of the French population (Mucchielli, 2009), the vulnerability dynamics laid out here propose that the actions of the French government significantly reinforced the riots. Overall, the explained interplay between pressure, mistakes, and becoming vulnerable, followed by undesired events which need to be addressed describe a first step in building a theory of vulnerability dynamics. Table 7 summarises testable propositions based on the previous explanation.

Table 7: Summary of Propositions for Vulnerability Dynamics

1	Trading off different objectives in an environment characterised by resource constraints leads to pressure.
2	To overcome pressure, fast actions are chosen which initially decrease pressure but also increase the number of mistakes, finally making oneself vulnerable in the eyes of an external actor.
3	Vulnerabilities increase the likelihood of undesired events as an external actor may use the weakness.
4	The exploitation of vulnerabilities describes and escalatory pattern as only demotivation may stop an external actor from further undesired events.
5	Undesired events cause reactive actions in an effort to decrease the probability of future events.
6	Through addressing mistakes, vulnerabilities and reacting on undesired events, less time and fewer resources are available for the actual tasks and objectives, reinforcing pressure.

Finally, next to the six previous contributions and the large range of theoretical and practical implications for the field of secure software development and cyber security, this study provides as seventh and last contribution a general theory on vulnerability dynamics. Due to its intended simplicity (Pidd, 2003) this theory does not aim to serve as only explanation for complex phenomena, such as software vulnerabilities and cyber attacks or social disadvantages and demonstrations, but rather to enrich the discussion about those by a dynamic perspective. Table 8 summarises the seven contributions of this study.

Table 8: Summary of Contributions of this Study

1	Provide a rich description of the interaction between pressure in software development, software vulnerabilities, the malicious interference from external cyber attacks, and organisational attack mitigation.
2	Describe a pathway to exploitation by explicitly connecting pressure, defects, and vulnerabilities.
3	Explain the dilemma between fixing vulnerabilities fast to avoid successful exploitation and potential problems arising from firefighting due to fast problem solving.
4	See cyber adversaries as regular competitors and integrate business, ICT, and cyber security strategies.
5	Explain a mechanism of dual firefighting when addressing both software vulnerabilities and cyber attacks.
6	Show that initial short-term gains from from functionality may be lost due to long-term insecurity.
7	Provide necessary conditions and testable propositions for a theory on vulnerability dynamics.

5.3 Limitations and Future Research

Paraphrasing Rahmandad and Repenning (2016), the findings of this research come with the usual caveats of single case studies. There are three aspects that affect the validity of this study in particular. First, there may be other, rival explanations (II.B, C, D) for growing numbers of software vulnerabilities and successful cyber attacks, such as the rising importance of technology (Goodman, 2016), and dependencies between teams and software (II.C; II.D). Although technology and dependencies are plausible explanations, they have been deliberately excluded from this study to keep the investigation focused. Future research could use the findings of this study as a starting point for an integrated socio-technical investigation to deepen the understanding of software vulnerabilities and successful cyber attacks. Second, relying on the qualitative data from one organisation impedes the generalisability of the study's findings and was further impaired by not having being able to record workshops or interviews (Yin, 2014). The study addressed these threats to validity by relying on several data sources to triangulate the findings, by employing several data analysis techniques and employing causal models to strengthen the analysis, by continuously discussing and deliberately disconfirming the findings with expert in the European financial organisation, and by comparing the insights with theory from literature as common in case study research (Thurmond, 2001; Yin, 2014). Additionally, previous research has discussed and proven the value of qualitative case studies in system dynamics research (e.g., Burchill, & Fine, 1997; Coyle, 2000, 2001; Repenning, & Sterman, 2002). Yet, these limitations offer two opportunities for future research: On the one hand, further case studies could be done in the same or in a different industry to validate, compare, and assess the findings from this research. On the other hand, future studies could use a quantitative system dynamics model to assess the testable propositions of this study through scenario analysis. Third, despite addressing the subject of benefits and risks, this study does not explicitly consider financial or economic implications. Instead, this study was based on the assumptions that ICT improves performance and that software vulnerabilities will lead to disproportionate costs. While both assumptions are deeply grounded in previous studies (e.g., Anderson et al., 2013; Wade, & Hulland, 2004),, future research could take this as a starting point to investigate the trade-off between software functionality and software security from a financial perspective through the lens of security economics and thereby provide clear and valuable decision support for practice.

REFERENCES

Scientific References, Books, Reports and Documentaries

Ablon, L., & Bogart, A. (2017). *Zero days, thousands of nights: The life time of zero-day vulnerabilities and their exploits.* Santa Monica, CA: RAND Corporation.

Ahmad, A., Maynard, S. B., & Park, S. (2014). Information security strategies: Towards an organizational multi-strategy perspective. *Journal of Intelligent Manufacturing, 25*(1), 357-370.

Ahmad, D. (2007). The contemporary software security landscape. *IEEE Security & Privacy*, 5(3), 75-77.

Amit, R., & Zott, C. (2001). Value creation in e-business. *Strategic Management Journal, 22*, 493-520.

Andersen, D. F., & Richardson, G. P. (1997). Scripts for group model building. System Dynamics Review, 13(2), 107-129.

Anderson, R., Barton, C., Böhme, R., Clayton, R., Van Eeten, M. J., Levi, M., & Savage, S. (2013). Measuring the cost of cybercrime. In *The economics of information security and privacy* (pp. 265-300). Berlin & Heidelberg, Germany: Springer.

Anderson, R. A., Crabtree, B. F., Steele, D. J., & McDaniel Jr., R. R. (2005). Case study research: The view from complexity science. Qualitative Health Research, 15(5), 669-685.

Argyris, C. (1992). *On organizational learning.* Cambridge, UK: Blackwell.

Argyris, C., & Schön, D. A. (1978). *Organizational learning: A theory of action perspective.* Reading, MA: Addison-Wesley.

Arkin, B., Stender, S., & McGraw, G. (2005). Software penetration testing. IEEE Security & Privacy, 3(1), 84-87.

Arora, A., Caulkins, J. P., & Telang, R. (2006). Research Note - Sell first, fix later: Impact of patching on software quality. *Management Science, 52*(3), 465-471.

Austin, R. D. (2001). The effects of time pressure on quality in software development: An agency model. *Information Systems Research, 12*(2), 195-207.pdf

Azoulay, P., Repenning, N. P., & Zuckerman, E. W. (2010). Nasty, brutish, and short: Embeddedness failure in the pharmaceutical industry. *Administrative Quarterly Science, 55*(3), 472-507.

Barlas, Y. (1996). Formal aspects of model validity and validation in system dynamics. *System Dynamics Review, 12*(3), 183-210.

Bauer, J. M., & van Eeten, M. (2011). Introduction to the economics of cybersecurity. *Communications and Strategies, 81*(1), 13-21.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D., J., (2001). *Agile Manifesto.* Retrieved from http://agilemanifesto.org/

Becker, C. (2014). Sustainability and longevity: Two sides of the same quality? In *Proceedings of the Third International Workshop on Requirements Engineering for sustainable Systems at the 22nd IEEE International Requirements Engineering Conference*. Karlskrona, Sweden.

van den Belt, M. (2004). Mediated modeling: A system dynamics approach to environmental consensus building; Washington, DC: Island Press.

Bharadwaj, A. S. (2000). A resource-based perspective on information technology capability and firm performance: An empirical investigation. *MIS Quarterly, 24*(1), 169-196.

Black, L. (2002). *Collaborating across boundaries: Theoretical, empirical, and simulated explorations.* PhD Dissertation. Cambridge, MA: Sloan School of Management, MIT.

Black, L., Carlile, P., & Repenning, N. P. (2004). A dynamic theory of expertise and occupational boundaries in new technology implementation: Building on Barley's study of CT scanning. *Administrative Science Quarterly, 49*, 572-607.

Boehm, B. (1988). A spiral model of software development and enhancement. *Computer, 21*(5), 61-72.

Boehm, B. (1984). Software engineering economics. *IEEE Transactions on Software Engineering, 10*(1), 4-21.

Boehm, B., & Turner, R. (2005). Management challenges to implementing agile processes in traditional development organizations. *IEEE software, 22*(5), 30-39.

Böhme, R., & Moore, T. (2009). The iterated weakest link. In *Workshop on the Economics of Information Security (WEIS)* (pp. 25-53). London, UK.

Bojanc, R., & Jerman-Blazič, B. (2008). An economic modelling approach to information security risk management. *International Journal of Information Management, 28*, 413-422.

Braun, W. 2002. *The system archetypes.* Albany, NY: University at Albany. State University of New York. Retrieved from http://www.albany.edu/faculty/gpr/PAD724/724WebArticles/sys_archetypes.pdf

Broderick, J. S. (2001). Information security risk management: When should it be managed? *Information Security Technical Report, 6*(3), 12-18.

Brooks, F. R. (1990). *The mythical man month* (2nd Ed.). Boston, MA: Addison Wesley.

Brynjolfsson, E., & Hitt, L. M. (2000). Beyond computation: Information technology, organizational transformation and business performance. *Journal of Economic Perspectives, 14(*4), 23-48.

Burchill, G., & Fine, C. H. (1997). Time versus market orientation in product concept development: Empirically-based theory generation. *Management Science, 43*(4), 465-478.

Cabinett Office (2010a). *Strategic framework and policy statement on improving the resilience of critical infrastructure to disruption from natural hazards.* London, UK.

Cabinett Office (2010b). *Sector resilience plan for critical infrastructure 2010.* London, UK.

Cao, L., Ramesh, B., Abdel-Hamid, T. (2010). Modelling dynamics in agile software development. *ACM Transactions on Management Information Systems, 1*(1), Article 5.

Collis, D. J. (1994). Research note: How valuable are organisational capabilities? *Strategic Management Journal, Winter Special Issue 15*(2), 143-152.

Colquitt, J. A., & Zapata-Phelan, C. P. (2007). Trends in theory building and theory testing: A five-decade study of the Academy of Management Journal. *Academy of Management Journal, 50*(6), 1281-1303.

Cosenz, F., & Noto, G. (2016). Applying system dynamics modelling to strategic management: A literature review. *Systems Research and Behavioral Science, 33*, 703-741.

Coyle, G. (2001). Rejoinder to Homer and Oliva. *System Dynamics Review, 17*, 357-363.

Coyle, G. (2000). Qualitative and quantitative modelling in System Dynamics: Some research questions. *System Dynamics Review, 16*(3), 225-244.

Davis, J. P., Eisenhardt, K. M., & Bingham, C. B. (2007). Developing theory through simulation methods. *Academy of Management Review, 32*(2), 480-499.

Dapp, T. F. (2014). Fintech: The digital (r)evolution in the financial sector. *Deutsche Bank Research.* Frankfurt am Main, Germany. Retrieved from http://dbresearch.com/PROD/DBR_INTERNET_DE-PROD/PROD0000000000345837/Fintech+%E2%80%93+The+digital+(r)evolution+in+the+financia.pdf

Deloitte (2016). *Cyber Value at Risk in the Netherlands.* Retrieved from https://www2.deloitte.com/content/dam/Deloitte/nl/Documents/financial-services/deloitte-nl-fsi-cyber-value-at-risk.pdf

De Nederlandsche Bank (2016). *Overview of financial stability: Spring 2016.* Amsterdam, the Netherlands.

De Nederlandsche Bank (2015). *Overview of financial stability: Spring 2015*. Amsterdam, the Netherlands.

Dierickx, I., & Cool, K. (1989). Asset stock accumulation and sustainability of competitive advantage. *Management Science, 35*(12), 1504-1511.

Diker, V. G. (2003). *Toward a dynamic feedback theory of open online collaboration communities.* PhD Dissertation. Albany, NY: University at Albany, The State University of New York.

DNI (2012). *Global trends 2030: Alternative worlds.* Washington D.C.: Office of the Director of National Intelligence. Retrieved from https://www.dni.gov/files/documents/GlobalTrends_2030.pdf

Dutta, A., & Roy, R. (2008). Dynamics of organizational information security. *System Dynamics Review, 24*, 349-375.

Eisenhardt, K. M., & Martin, J. A. (2000). Dynamic capabilities: What are they? *Strategic Management Journal, 21*(10-11), 1105-1121.

Eisenhardt, K. M., & Zbaracki, M. J. (1992). Strategic decision making. *Strategic Management Journal, 13*, 17-37.

Ethiraj, S. K., Kale, P., Krishnan, M. S., & Singh, J. V. (2005). Where do capabilities come from and how do they matter? A study in the software services industry. *Strategic management journal*, *26*(1), 25-45.

Forrester, J. W. (1992). Policies, decisions, and information sources for modeling. *European Journal of Operational Research, 59*, 42-63.

Forrester, J. W. (1971). Counterintuitive behavior of social systems. *Technology Review, 73*(3), 52-68.

Forrester, J. W. (1961). *Industrial dynamics.* Cambridge, MA: MIT Press.

Forrester, J. W. (1958). Industrial dynamics: a major breakthrough for decision making. *Harvard Business Review, 18*(2), 243-267.

Forrester, J. W., & Senge, P. M. (1980). Tests for building confidence in system dynamics. models. *TIMS Studies in Management Sciences*, *14*, 209-228.

Furnell, S., & Thomson, K. L. (2009). Recognising and addressing 'security fatigue'. *Computer Fraud and Security, 11*, 7-11.

Gary, M. S., Kunc, M., Morecroft, J. D. W., Rockart, S. F. (2008). System dynamics and strategy. *System Dynamics Review, 24*, 407-429.

Gill, P., Stewart, K., Treasure, E., & Chadwick, B. (2008). Methods of data collection in qualitative research: Interviews and focus groups. *British Dental Journal, 204,* 291-295.

Gillet, R., Hübner, G., & Plunus, S. (2010). Operational risk and reputation in the financial industry. *Journal of Banking & Finance, 34*, 224-235.

Goh, Y. M., Love, P. E. D., Brown, H., & Spickett, J. (2012). Organizational accidents: A systemic model of production versus protection. *Journal of Management Studies, 49*(1), 52-76.

Gonçalves, P., Hines, J., Sterman, J. D. (2005). The impact of endogenous demand on push-pull production systems. *System Dynamics Review, 21*, 187-216.

Goodman, M. (2016). *Future crimes: Inside the digital underground and the battle for our connected world.* London, UK: Corgi Books.

Gordon, L. A., & Loeb, M. P. (2002). The economics of information security investment. *ACM Transactions on Information Security and System Security, 5*(4), 438-457.

Grant, R. M. (2010). *Contemporary strategy analysis* (7th ed.). Chichester, UK: John Wiley & Sons.

Hamid, B., Gürgens, S., & Fuchs, A. (2016). Security patterns modeling and formalization for pattern-based development of secure software systems. *Innovations in Systems and Software Engineering, 12*, 109-140.

Harrison, J . R., Lin, Z., Carroll, G. R., & Carley, K. M. (2007). Simulation modelling in organizational and management research. *Academy of Management Review, 32*(4), 1229-1245.

Heitzenrater, C., Böhme, R., & Simpson, A. (2016). The days before Zero Day: Investment Models for Secure Software Engineering. *WEIS 2016.* Retrieved from http://weis2016.econinfosec.org/wp-content/uploads/sites/2/2016/05/WEIS_2016_paper-_21-2.pdf

Henderson, J. C., & Venkatraman, N. (1993). Strategic alignment: Leveraging information technology for transforming organzisations. I*BM Systems Journal, 32*(1), 472-484.

Hesseling, R. (1994). Displacement: A review of the empirical literature. *Crime prevention studies*, *3*(1), 197-230.

Homer, J. (1996). Why we iterate: Scientific modeling in theory and practice. *System Dynamics Review, 12*(1), 1-19.

Homer, J., & Oliva, R. (2001). Maps and models in System Dynamics: A response to Coyle. *System Dynamics Review, 17*(4), 347-355.

Huang, C. D., Hu, Q., & Beharam R. S. (2008). An economic analysis of the optimal information security investment in the case of a risk-averse firm. *International Journal of Production Economics, 114*, 793-804.

Johnson, S. D., Guerette, R. T., & Bowers, K. (2014). Crime displacement: what we know, what we don’t know, and what it means for crime reduction. *Journal of Experimental Criminology, 10*, 549-571.

Johnston, H. R., & Carrico, S. R. (1988). Developing capabilities to use information strategically. *MIS Quarterly, 12*(1), 37-48.

Kettinger, W. J., Grover, V., Guha, S., & Segars, A. H. (1994). Strategic information systems revisited: A study in sustainability and performance. *MIS Quarterly, 18*(1), 31-58.

Kissel, R., Stine, K., Scholl, M., Rossman, H., Fahlsing, J., & Gulick, J. (2008). Security considerations in the system development lifecycle: Information security. *NIST Special Publications 800-64 Revision 2*. Gaithersburg, MD: National Institute of Standards and Technology, U.S. Department of Commerce.

Von Kogh, G., Rossi-Lamastra, C., & Haefliger, S. (2012). Phenomenon-based research in management and organisation science: When is it rigorous and does it matter? *Long Range Planning, 45*, 277-298.

Kopainsky, B., & Luna-Reyes, L. F. (2008). Closing the loop: Promoting synergies with other theory building approaches to improve system dynamics practice. *Systems Research and Behavioral Science, 25*, 471-486.

Landwehr, C. E. (2001). Computer security. *International Journal of Information Security, 1*, 3-13.

Laverty, K. J. (1996). Economic “short-termism”: The debate, the unresolved issue, and the implications for management practice and research. *Academy of Management Review, 21*(3), 825-860.

Leopold, H., Bleier, T., & Skopik, F. (2015). *Cyber Attack Information System: Erfahrungen und Erkenntnisse aus der IKT-Sicherheitsforschung.* Berlin, Heidelberg, Germany: Springer.

Levinthal, D. A., & March, J. G. (1993). The myopia of learning. *Strategic Management Journal, Winter Special Issue 14*, 95-112.

Libicki, M. C., Ablon, L., & Webb, T. (2015). *The defender’s dilemma: Charting a course toward cybersecurity.* Santa Monica, CA: RAND Corporation.

Luna-Reyes, L. F. (2004). *Collaboration, trust and knowledge sharing in information-technology-intensive projects in the public sector.* PhD Dissertation. Albany, NY: University at Albany, The State University of New York.

Luna-Reyes, L. F., & Andersen, D. L. (2003). Collecting and analyzing qualitative data for system dynamics: methods and models. *System Dynamics Review, 19*, 271-296.

Luna-Reyes, L. F., Martinez-Moyano, I. J., Pardo, T. A., Cresswell, A. M., Andersen, D. F., & Richardson, G. P. (2006). Anatomy of group-model building intervention: Building dynamic theory from case study research. *System Dynamics Review, 22*, 291-320.

Lupien, S. J., Maheu, F., Tu, M., Fiocco, A., & Schramek, T. E. (2007). The effects of stress and hormones on human cognition: Implications for the field of brain and cognition. *Brain and Cognition, 65*, 209-237.

Martinez-Moyano, I. J., Conrad, S. H., & Andersen, D. F. (2011). Modeling behavioral considerations related to information security. *Computers and Security, 30*, 397-409.

Martinez-Moyano, I. J., McCaffrey, D. P., & Oliva, R. (2014). Drift and adjustment in organizational rule compliance: Explaining the regulatory pendulum in financial markets. *Organization Science, 25*(2), 321-338.

Martinez-Moyano, I. J., Oliva, R., Morrison, D., & Sallach, D. (2015). Modeling adversarial dynamics. In *Proceedings of the 2015 Winter Simulation Conference* (pp. 2412-2423). IEEE Press.

Martinez-Moyano, I. J., Rich, E., Conrad, S., Andersen, D. F., & Stewart, T. R. (2008). A behavioral theory of insider-threat risks: A system dynamics approach. *ACM Transactions on Modeling and Computer Simulation*, *18*(2), 7.

MacCormack, A., Kemerer, C. F., Cusumano, M., & Crandall, B. (2003). Trade-offs between productivity and quality in selecting software development practices. *IEEE Software, 20*(5), 78-85.

McAfee (2014). *Net losses: Estimating the global costs of cybercrime.* Retrieved from https://www.mcafee.com/us/resources/reports/rp-economic-impact-cybercrime2.pdf

McAfee (2013). *The economic impact of cybercrime and cyber espionage.* Retrieved from https://www.mcafee.com/uk/resources/reports/rp-economic-impact-cybercrime.pdf

McGraw, G. (2006). *Software security: Building security in.* Boston, MA: Addison-Wesley.

McGraw, G., Migues, S., & West, J. (2016). *BSIMM 7.* Dulles, VA: Cigital.

Meadows, D. H. (2009). *Thinking in systems: A primer.* London, UK: Earthscan.

Miller B., & Rowe D. (2012). A survey SCADA of and critical infrastructure incidents. In: RIIT, *Proceedings of the 1st annual conference on research in information technology* (pp. 51-56). New York City, NY: ACM.

Mohammed, N. M., Niazi, M., Alshayeb, M., & Mahmood, S. (2017). Exploring software security approaches in software development lifecycle: A systematic mapping study. *Computer Standards & Interfaces, 50*, 107-115.

Morecroft, J. D. W. (1991). Executive knowledge, models, and learning. *European Journal of Operational Research, 59*, 9-27.

Morecroft, J. D. W., Lane, D. C., & Viita, P. S. (1991). Modeling growth strategy in a biotechnology startup firm. *System Dynamics Review, 7*(2), 93-116.

Mucchielli, L. (2009). Autumn 2005: A review of the most important riot in the history of French contemporary society. *Journal of Ethnic and Migration Studies, 35*(5), 731-735.

National Cyber Security Centre (2016). *Cyber security assessment Netherlands: CSAN 2016.* Den Haag, the Netherlands.

NIST (2014). *Framework for improving critical infrastructure cybersecurity.* Geithersburg, MD: National Institute for Standards and Technology.

Neumann, P. G. (2012). Inside Risks: The foresight saga, redux: Short-term thinking is the enemy of the long-term future. *Communications of the ACM, 55*(10), 26-29.

Nutt, P. C. (2002). *Why decision fail: Avoiding the blunders and traps that lead to debacles.* San Francisco, CA: Barrett-Koehler.

Oliva, R., & Sterman, J. D. (2001). Cutting corners and working overtime: Quality erosion in the service industry. *Management Science 47*(7), 894–914.

van Oorschot, K. E., Akkermans, H., Sengupta, K., van Wassenhove, L. N. (2013). Anatomy of a decision trap in complex new product development projects. *Academy of Management Journal, 56(*1), 285-307.

OWASP (2013). *OWASP Top 10 2013: The ten most critical web application security risks.* Retrieved from https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/owasptop10/OWASP%20Top%2010%20-%202013.pdf

Perlow, L. A., Okhuysen, G., & Repenning, N. P. (2002). The speed trap: Exploring the relationship between decision making and temporal context. *Academy of Management Journal, 5*, 931-955.

Perlow, L. A., & Repenning, N. P. (2009). The dynamics of silencing conflict. *Research in Organizational Behaviour, 29*, 195-223.

Pidd, M. (2003). *Tools for thinking: Modelling in management science* (2nd ed.). Chichester, UK: John Wiley and Sons.

Piessens, F. (2002). A taxonomy of causes of software vulnerabilities in internet software. In *Supplementary Proceedings of the 13th International Symposium on Software Reliability Engineering* (pp. 47-52). Los Alamitos, CA: IEEE Computer Society Press.

Pfleeger, C. P., Pfleeger, S. L., & Margulies, J. (2015). *Security in computing* (5th ed.). Upper Saddle River, NJ: Prentice Hall.

Ponemon Institute (2016). *2016 costs of cyber crime study and the risk of business innovation.* Retrieved from http://www.ponemon.org/local/upload/file/2016%20HPE%20CCC%20GLOBAL%20REPORT%20FINAL%203.pdf

Porter, M. E. (1991). Towards a dynamic theory of strategy. *Strategic Management Journal, Winter Special Issue,12*(52), 95-117.

Porter, M. E. (1979). How competitive forces shape strategy. *Harvard Business Review, 57*(2), 137-145.

Porter, M. E., & Millar, V. E. (1985). How information gives you competitive advantage. *Harvard Business Review, 63*(4), 149-160.

Powell, T. C., & Dent-Micallef, A. (1997). Information technology as competitive advantage: The role of human business, and technology resources. *Strategic Management Journal, 18*(5), 375-405.

Pressman, R. S. (2010). *Software engineering: A practitioners approach* (7th ed.). Boston, MA: McGraw-Hill.

Rahmandad, H. (2012). Impact of growth opportunities and competition on firm-level capability development trade-offs. *Organization Science, 23*(1), 138-154.

Rahmandad, H. (2005). *Three essays on modeling dynamic organizational processes.* PhD Dissertation. Cambridge, MA: Sloan School of Management, MIT.

Rahmandad, H., Henderson, R., & Repenning, N. P., (2016). Making the numbers? “Short Terminism” and the puzzle of occasional disaster. *Management Science*, 1-21.

Rahmandad, H. & Hu, K. (2010). Modelling the rework cycle: Capturing multiple defects per task. *System Dynamics Review, 26*, 291-315.

Rahmandad, H., & Repenning, N. P. (2016). Capability erosion dynamics. *Strategic Management Journal, 37*, 649-672.

Ravichandran, T. & Lertwongsatien, C. (2005). Effect of information systems resources and capabilities on firm performance: A resource-based perspective. *Journal of Management Information Systems, 21*(4), 237-276.

Repenning, N. P. (2003). Selling system dynamics to (other) social scientists. *System Dynamics Review, 19*, 303-327.

Repenning, N. P. (2001). Understanding fire fighting in new product development. *The Journal of Product Innovation Management, 18*(5), 285-300.

Repenning, N., Gonçalves, P., Black, L., (2001). Past the tipping point: The persistence of fire fighting in product development. *California Management Review 43*(4): 44–63.

Repenning, N. P., & Sterman, J. D. (2002). Capability traps and self-confirming attribution errors in the dynamics of process improvement. *Administrative Science Quarterly, 47*(2), 265-295.

Richardson, G. P. (2013). Concept models in group model building. *System Dynamics Review, 29*(1), 42-55.

Rivard, S., Raymond, L., & Verreault, D. (2006). Resource-based view and competitive strategy: An integrated model of the contribution of information technology to firm performance. J*ournal of Strategic Information Systems, 15*, 29-50.

Rouwette, E. A. J. A., & Franco, L. A. (Unpublished). *Messy problems: Practical interventions for working through complexity, uncertainty and conflict.*

Rudolph, J. W., Morrison, J. B., & Carroll, J. S. (2009). The dynamics of action-oriented problem solving: Linking interpretation and choice. *Academy of Management Review, 34*(4), 733-756.

Rudolph, J. W. & Repenning, N. P. (2002). Disaster dynamics: Understanding the role of quantity in organizational collapse. *Administrative Science Quarterly, 47*, 1-30.

Sastry, M. A. (1997). Problems and paradoxes in a model of punctuated organizational change. *Administrative Science Quarterly, 42*, 237-275.

Schneider, C. L. (2011). Violence and State Repression. *Swiss Political Science Review, 17*(4), 480-484.

Schwaber, K. (2004). *Agile project management with Scrum.* Redmond, WA: Microsoft Press. (EPUP 13 inch monitor)

Scott, R. J., Cavana, R. Y., & Cameron, D. (2015). Recent evidence on the effectiveness of group model building. *European Journal of Operational Research, 249*, 908-918.

Shumba, R., Walden, J., Ludi, S., Taylor, C., An Wang, A. J. (2006). Teaching the secure software development lifecycle: Challenges and experiences. In *Proceedings of the 10th Colloquium for Information Systems Security Education* (pp. 116-123). Adelphi, MD: University of Maryland.

Simon, H. A. (1985). Human nature in politics: The dialogue of psychology with political science. *The American Political Science Review, 79*(2), 293-304.

von Solms, R., & van Niekerk, J. (2013). From information security to cyber security. *Computers & Security, 38*, 97-102.

Stecklein, J. M., Dabney, J., Dick, B., Haskins, B., Lovell, R., & Moroney, G. (2004). Error cost escalation through the project lifecycle. *In 14th Annual International Symposium of INCOSE*.

Sterman, J. D. (2006). Learning from evidence in a complex world. American Journal of Public Health, 96(3), 505-514.

Sterman, J. D. (2002). All models are wrong: Reflections on becoming a systems scientist. *System Dynamics Review, 18*(4), 501-531.

Sterman, J. D. (2000). *Business dynamics: System thinking and modelling for a complex world.* Boston, MA: McGraw-Hill.

Sterman, J. D. (1994). Learning in and about complex systems. *System Dynamics Review, 10*(2-3), 291-330.

Strauss, A., & Corbin, J. (1994). Grounded theory methodology: An overview. In N. K. Denzin, & Y. S. Lincoln (Eds.), *Handbook of qualitative research* (pp. 273-285). Thousand Oaks, CA: Sage Publications.

Sutton, R. I., & Staw, B. M. (1995). What theory is not. *Administrative Science Quarterly, 40*(3), 371-384.

Teece, D. J., Pisano, G., & Shuen, A. (1997). Dynamic capabilities and strategic management. *Strategic Management Journal, 18*(7), 509-533.

Telang, R., & Wattal, S. (2007). An empirical analysis of the impact of software vulnerability announcements on firm stock price. *IEEE Transactions on Software Engineering*, *33*(8), 544-557.

Teleb, C. W., Weisburd, D., Gill, C. E., Vitter, Z., & Teichman, D. (2014). Displacement of crime and diffusion of crime control benefits in large-scale geographic areas: A systematic review. *Journal of Experimental Criminology, 10*, 515-548.

Thurmond, V. A. (2001). The point of triangulation. *Journal of Nursing Scholarship, 33*(3), 253-258.

Tversky, A., & Kahneman, D. (1986). Rational choice and the framing of decision. *Journal of Business, 59*(4) pt 2, 251-278.

Tversky, A., & Kahneman, D. (1974). Judgement under uncertainty. *Science, 185*(4157), 1124-1131.

Vaughan, D. (1992). Theory elaboration: The heuristics of case analysis. In H. Becker, & C. Ragin (Eds.), *What is a case?* (pp. 173-202.) Cambridge, UK: Cambridge University Press.

Vennix, J. A. M. (1996). *Group model building: Facilitating team learning using system dynamics.* Chichester, UK: John Wiley and Sons.

Vennix, J. A. M., Andersen, D. F., Richardson, G. P., & Rohrbaugh, J. (1992). Model-building for group decision support: issues and alternatives in knowledge elicitation. *European Journal of Operational Research, 59,* 28-41.

Verizon (2016). *2016 Data Breach Investigations Report.* Basking Ridge, NY. Retrieved from http://www.verizonenterprise.com/resources/reports/rp_DBIR_2016_Report_en_xg.pdf

Videira, N., Antunes, P., & Santos, R. (2017). Engaging stakeholders in environmental and sustainability decisions with Participatory System Dynamics Modeling. In S. Gray, M. Paolisso, R. Jordan, & S. Gray (Eds.), *Environmental Modeling with Stakeholders* (pp. 241- 268. Cham, Switzerland: Springer International Publishing.

Wade, M. & Hulland, J. (2004). Review: The resource-based view and information systems research: Review, extension, and suggestions for future research. *MIS Quarterly, 28*(1), 107-142.

Walrave, B., van Oorschot, K. E., & Romme, A. G. L. (2011). Getting trapped in the suppression of exploration: A simulation model. *Journal of Management Studies, 48*(8), 1727-1751.

Wheat, D. (2015). Model-based policy design that takes implementation seriously. In E. Johnston (Eds.), *Governance in the information era* (pp. 101-118). New York City, NY: Routledge.

Wheat, D. (2010). What can system dynamics learn from the public policy implementation literature? *Systems Research and Behavioral Science, 27*, 425-442.

Widman, J., Hua, S. Y., & Ross, S. C. (2010). Applying lean principles in software development process: A case study. *Issues in Information Systems*, *9*(1), 635-639.

De Win, B., Scandariato, R., Buyens, K., Grégoire, J., & Joosen, W. (2009). On the secure software development process: CLASP, SDL, and touchpoints compared. *Information and Software Technology, 51*, 1152-1171.

Winter, S. G. (2003). Understanding dynamic capabilities. *Strategic Management Journal, 24*(10), 991-995.

Wysopal, C. (2012). Software security varies greatly. *Datenschutz und Sicherheit, 9*, 645-652.

Yerkes, R. M., & Dodson, J. D. (1908). The relation of strength of stimulus to rapidity of habit formation. *Journal of Comparative Neurology and Psychology, 18*, 459–482.

Yin, R. K. (2014). Case study research: Design and methods (5th ed.). Thousand Oaks, CA: Sage Publications.

Zagonel, A. A. (2002). Model conceptualization in group model building: A review of the literature exploring the tension between representing reality and negotiating a social order. In *Proceedings of the 2002 international conference of the system dynamics society*. Albany, NY: System Dynamics Society.

References from Newspapers, Webpages, or Blogs

Adobe (2016). *Security in Engineering: Building Security into our products and services.* Retrieved from https://www.adobe.com/security/engineering.html

Arthur, C. (2011, September 5). DigiNotar SSL certificate hack amounts to cyberwar, experts says. *The Guardian*. Retrieved from https://www.theguardian.com/technology/2011/sep/05/diginotar-certificate-hack-cyberwar

Chrisafis, A. (2015, October 22). Nothing's changed: 10 years after French riots, banlieues remain in crisis. *The Guardian.* Retrieved from https://www.theguardian.com/world/2015/oct/22/nothings-changed-10-years-after-french-riots-banlieues-remain-in-crisis

Crowe, P. (2016, March 30). CITI: The 'Uber moment' for banks is coming — and more than a million people could lose their jobs. *Business Insider.* Retrieved from http://www.businessinsider.com/bank-layoffs-are-coming-2016-3?international=true&r=US&IR=T

CVE (2017). Common vulnerabilities and exposures: The standard for information security vulnerability names. Retrieved from: https://cve.mitre.org/

FinExtra (2017, March 9). DNB says its future is to become a "technology company with a banking licence". *FinExtra.* Retrieved from https://www.finextra.com/newsarticle/

30250/dnb-says-its-future-is-to-become-a-technology-company-with-a-banking-license

Fox-Brewster, T. (2017, June 27). Petya or NotPetya: Why the latest ransomware is deadlier than WannaCry. Forbes. Retrieved from https://www.forbes.com/sites/thomasbrewster/2017/06/27/petya-notpetya-ransomware-is-more-powerful-than-wannacry/#36f7b4e2532e

Gandel, S. (2016, June 27). Here's how Citigroup is embracing the 'Fintech' revolution. Fortune. Retrieved from http://fortune.com/citigroup-fintech/

The Hague Security Delta (2017). The Hague Security Delta: Leading security cluster in Europe. *The Hague Security Delta.* Retrieved from https://www.thehaguesecuritydelta.com/

Lopez, L. (2013, February 27). Here's why Wall Streeters are STILL getting axed. *Business Insider.* Retrieved from http://www.businessinsider.com/why-wall-street-bank-layoffs-wont-stop-2013-2?international=true&r=US&IR=T

Microsoft (2017a). *Security Development Lifecycle.* Retrieved from https://www.microsoft.com/en-us/SDL/process/training.aspx

Microsoft (2017b). *SDL for Agile.* Retrieved from https://www.microsoft.com/en-us/SDL/Discover/sdlagile.aspx

OWASP (2017, July 4). OWASP SAMM Project. Retrieved from https://www.owasp.org/index.php/OWASP_SAMM_Project

OWASP (2016, August 8). CLASP Concepts. OWASP. Retrieved from https://www.owasp.org/index.php/CLASP_Concepts

Rankin, J. (2013, May 28). 'Big four' banks cut 189,000 jobs worldwide in five years. *The Guardian.* Retrieved from https://www.theguardian.com/business/2013/may/28/big-four-banks-cut-jobs

Sciolino, E. (2007, November 29). Sarkozy pledges crackdown on rioters. The New York Times. Retrieved from http://www.nytimes.com/2007/11/29/world/europe/29-france.html

Zetter, K. (2011, September 20). DigiNotar files for bankruptcy in wake of devastating hack. *Wired*. Retrieved from https://www.wired.com/2011/09/diginotar-bankruptcy/

APPENDIX I - MODEL DOCUMENTATION

While the Results and Analysis Section (4) above depicts a highly aggregated causal diagram to address the research question, throughout the group model workshops several detailed models have been created, covering the areas of agile secure software development, defects and vulnerabilities, the DevOps, training and awareness, third party software, adversary behaviour, and responsible disclosure. Additionally, the submodels of those areas have been integrated to an overarching causal diagram.

This subsection portrays the iterative nature of creating a causal diagram. The diagrams generally indicate a final, an intermediate, and an initial version. Depending on the complexity, Occam's razor was applied to strip off all unnecessary details (Pidd, 2003). In this way the causal diagrams could be summarised, so that they are understandable to outsiders who did not take part in the workshops.

I. A Causal Diagrams Group Model Building Session 1

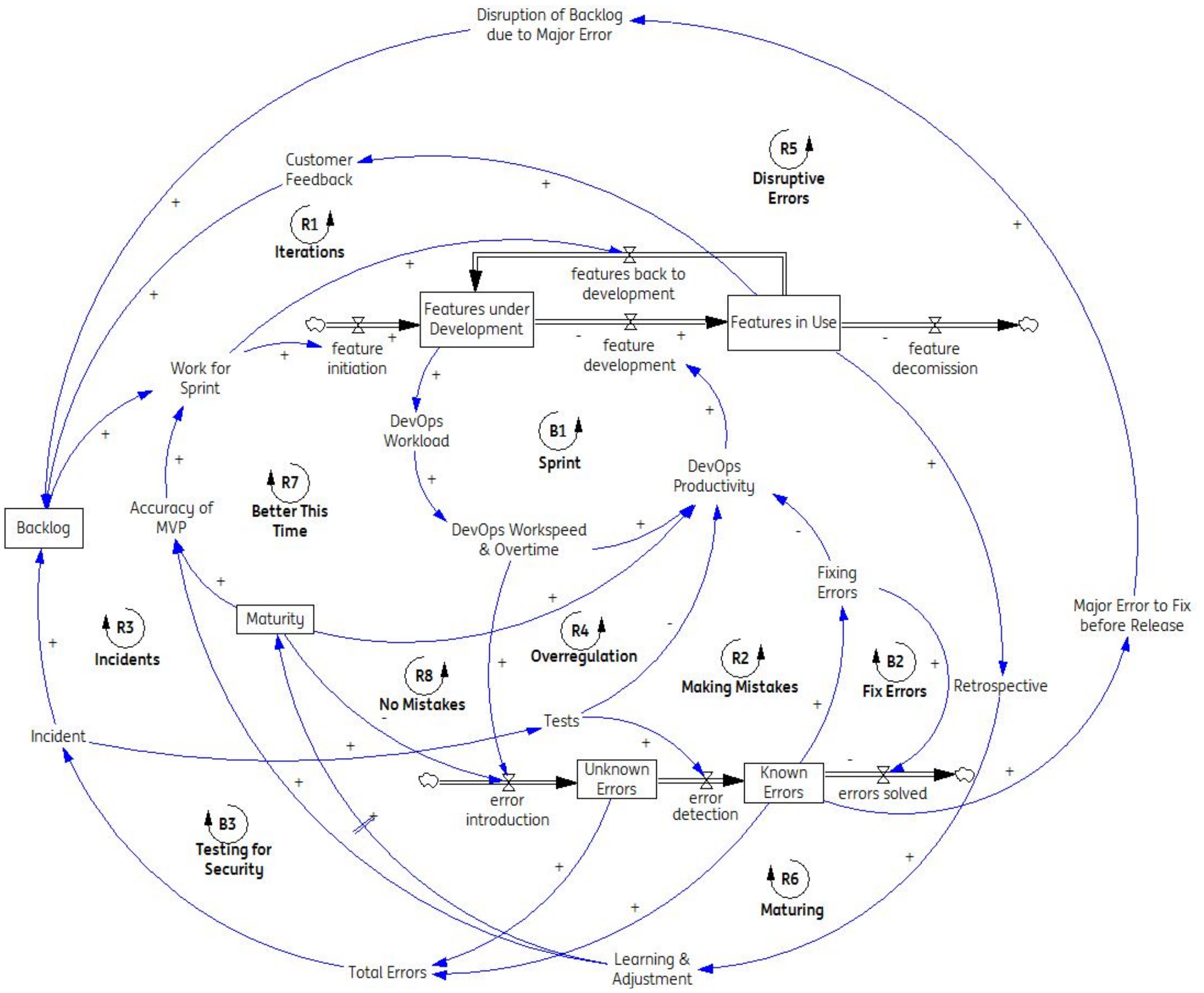


Figure I.A.1: Causal Diagram of Agile Secure Software Development, Defects, Backlog and Sprint Backlog, Incidents, and Maturity. This diagram represents the final version of the model created in the first workshop. The diagram was created by the researcher and validated by the participants.

As common in the financial organisation as well as in group model building, the results of the workshop were sent to the participants as a summary shortly after the session. In contrast to common practice in group model building (see e.g. Vennix, 1996), no workbooks were used because of the busy schedule of the participants. Instead, a short and comprehensive summary was created in powerpoint which is one of the major methods of conveying insights within the organisation. The powerpoint presentation simply depicted Figure I.A.1 above as final summarised version of the causal diagram of the first workshop and added the explanatory bullet point list below. Each point describes the dynamics of a feedback loop discussed within the session.

- B1: Software is developed.
- B2: Errors are fixed.
- B3: Tests reveal errors, leading to less incidents.
- R1: More customer feedback causes more iterations.
- R2: Higher work speed increases productivity but also creates errors.
- R3: More errors cause more incidents which in turn leads to new errors.
- R4: More incidents lead to more tests, decreasing productivity, finally causing new errors.
- R5: Major errors disrupt the planning, causing stress and finally new errors.
- R6: Via retrospective, successfully conducting sprints increases maturity.
- R7: Via retrospective, successfully conducting sprints increases accuracy in planning.
- R8: More maturity causes better software development.

The participants were invited to comment on the summary via e-mail, phone call, or in person prior to the second session. Additionally, the causal diagram shown in Figure I.A.1 was unfolded loop by loop in the second workshop and explained by the researcher. The participants were asked to correct and change the causal diagram according to their own ideas. When presenting the model, the researcher continuously probed the participants to disconfirm the causal diagram in order to increase its internal validity (Andersen et al., 2012; Yin, 2014). Despite the invitation to adjust the causal diagrams, the participants did not object to it, but instead, were very satisfied with the results.

The two causal diagrams below in Figure I.A.2 and Figure I.A.3 describe the intermediate and initial version of the model created during the first group model workshop.

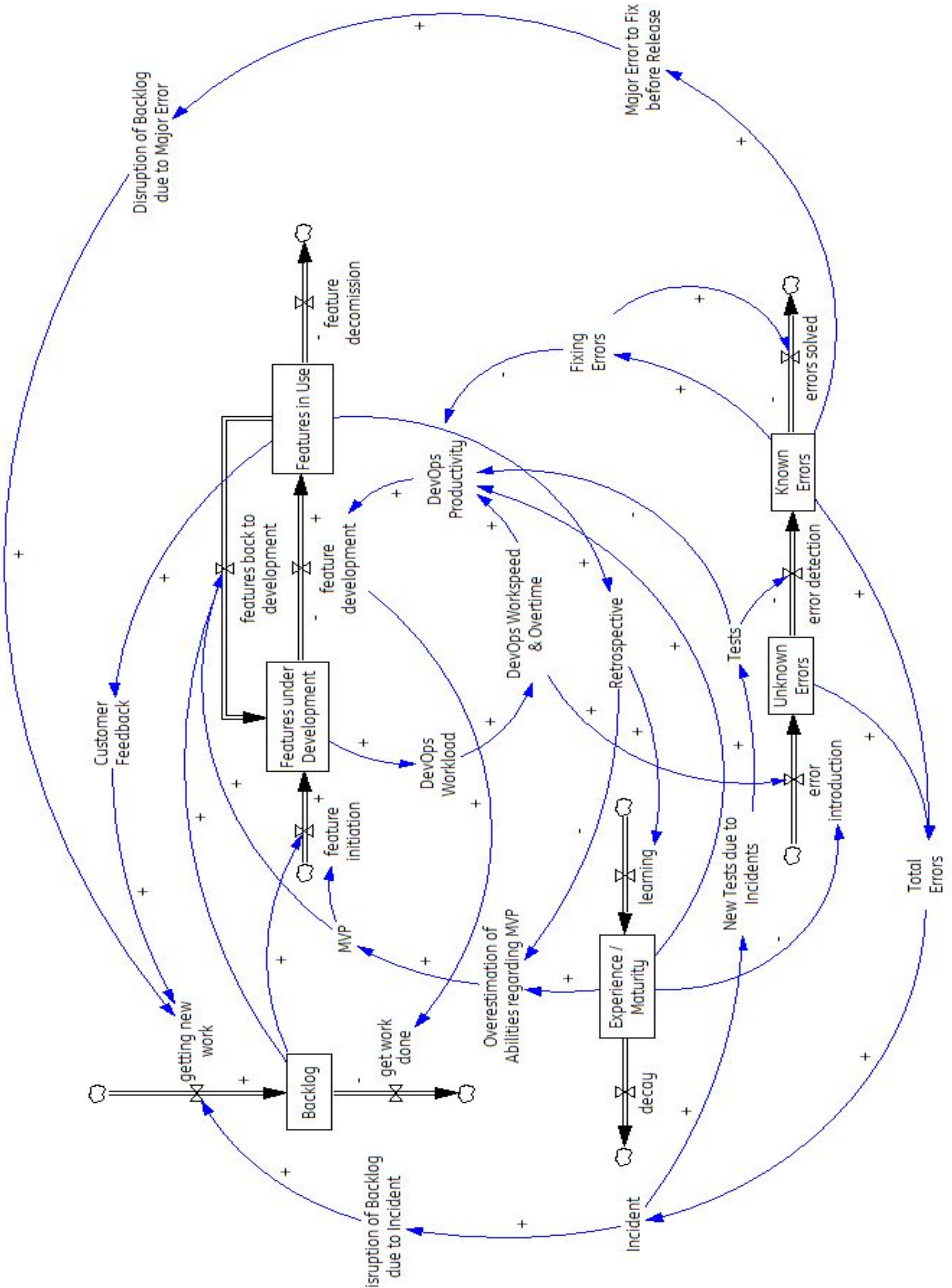


Figure I.A.2: Causal Diagram of Agile Secure Software Development, Defects, Backlog and Sprint Backlog, Incidents, and Maturity. This diagram represents an intermediated version of the model created in the first workshop. The diagram was created by the researcher, and discussed with a few participants who deemed it to be too complicated for further use. Together, it was decided to simplify the causal diagram further.

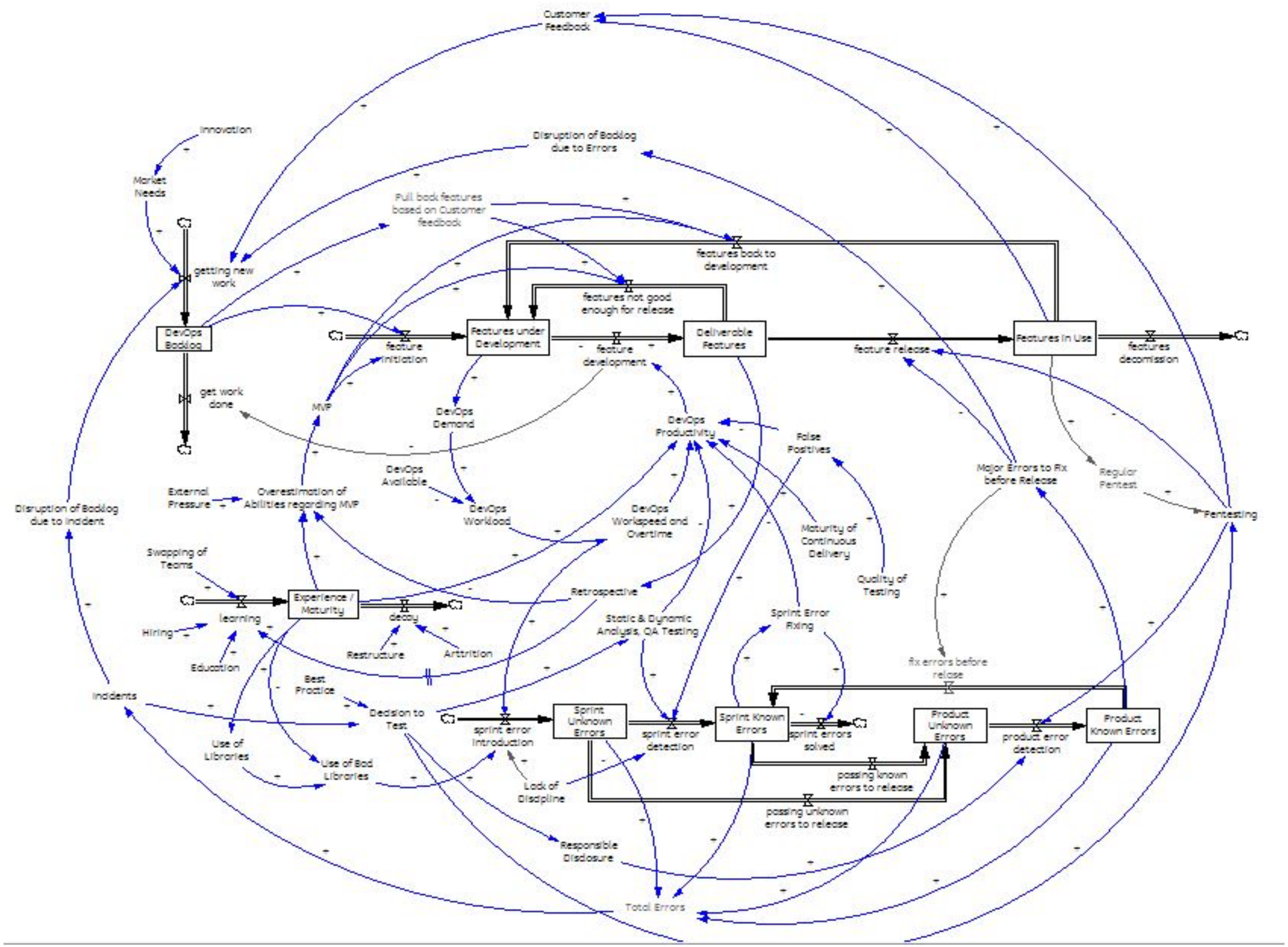


Figure I.A.3: Causal Diagram of Agile Secure Software Development, Defects, Backlog and Sprint Backlog, Incidents, and Maturity. This diagram represents the initial version of the model created in the first workshop. The diagram was created by the participants who drew it on paper. Afterwards, the researcher translated the diagram onto the computer but kept everything as it was decided by the participants. Since the participants were already laughing about the complexity of the diagram during the session, the researcher had decided to simplify the model, as eventually done in the figures above.

I. B Causal Diagrams Group Model Building Session 2

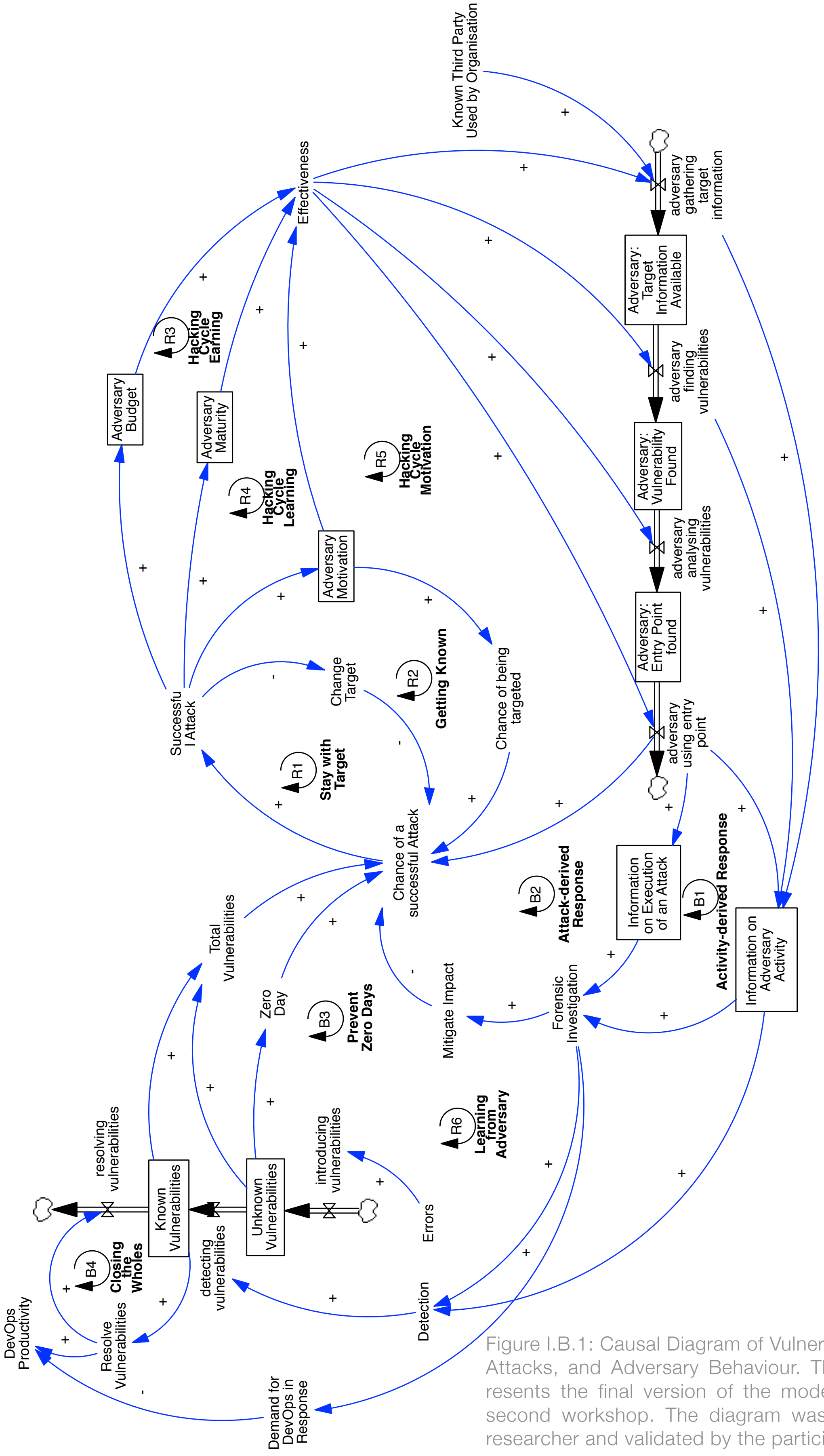

Known Third Party Used by Organisation
adversary gathering target information
Effectiveness
Adversary: Target Information Available
adversary finding vulnerabilities
Adversary: Vulnerability Found
adversary analysing vulnerabilities
Adversary: Entry Point found
adversary using entry point
Adversary Budget
R3 Hacking Cycle Earning
Adversary Maturity
R5 Hacking Cycle Motivation
R4 Hacking Cycle Learning
Adversary Motivation
Successfu l Attack
Change Target
R2 Getting Known
Chance of being targeted
R1 Stay with Target
Chance of a successful Attack
Information on Execution of an Attack
B2 Attack-derived Response
B1 Activity-derived Response
Information on Adversary Activity
Total Vulnerabilities
Zero Day
B3 Prevent Zero Days
Mitigate Impact
Forensic Investigation
R6 Learning from Adversary
resolving vulnerabilities
Known Vulnerabilities
Unknown Vulnerabilities
introducing vulnerabilities
Errors
B4 Closing the Wholes
detecting vulnerabilities
Detection
DevOps Productivity
Resolve Vulnerabilities
Demand for DevOps in Response

Figure I.B.1: Causal Diagram of Vulnerabilities, External Attacks, and Adversary Behaviour. This diagram represents the final version of the model created in the second workshop. The diagram was created by the researcher and validated by the participants.

Like with the first group model building session, the results of the second workshop were sent to the participants as a summary shortly after the session. Again, no workbooks were used because of the busy schedule of the participants. Instead, a short and comprehensive summary was created in powerpoint. The powerpoint presentation simply depicted Figure I.B.1 above and I.B.3 below as final summarised version of the causal diagrams of the second workshop. Additionally, the explanatory bullet point list below as well as the one following I.B.3 were added to the powerpoint presentation. Each point describes the dynamics of a feedback loop discussed within the session.

- B1: Defence actions based on information about adversary activities.
- B2: Defence actions based on information about adversary attack.
- B3: Detecting unknown vulnerabilities prevents zero days.
- B4: Vulnerabilities are resolved.

- R1: The less a hacker changes his/her target, the better they get to know it.
- R2: The larger the footprint of a target, the higher the motivation to attack it.
- R3: Successful attacks result in more budget, leading to further attack cycles.
- R4: Successful attacks result in more maturity, leading to further attack cycles.
- R5: Successful attacks result in more motivation, leading to further attack cycles.
- R6: Although always lagging behind, more attacker actions reveal more vulnerabilities.

Once more, the participants were invited to comment on the summary via e-mail, phone call, or in person prior to the third session. Like the second workshop, also the third sessions started with unfolding and explaining the causal diagrams depicted in the Figure I.B.1 and I.B.3 loop by loop. Again, the participants were requested to improve and adjust the causal diagrams according to their own ideas. When presenting the model, the researcher continuously probed the participants to disconfirm the causal diagrams in order to increase its internal validity (Andersen et al., 2012; Yin, 2014). Despite the invitation to adjust the causal diagrams, the participants did not object to it, but instead, were very satisfied with the outcomes.

The three causal diagrams below in Figure I.B.2, Figure I.B.4, and Figure I.B.5 show the intermediate and initial versions of the models created during the second group model building workshop.

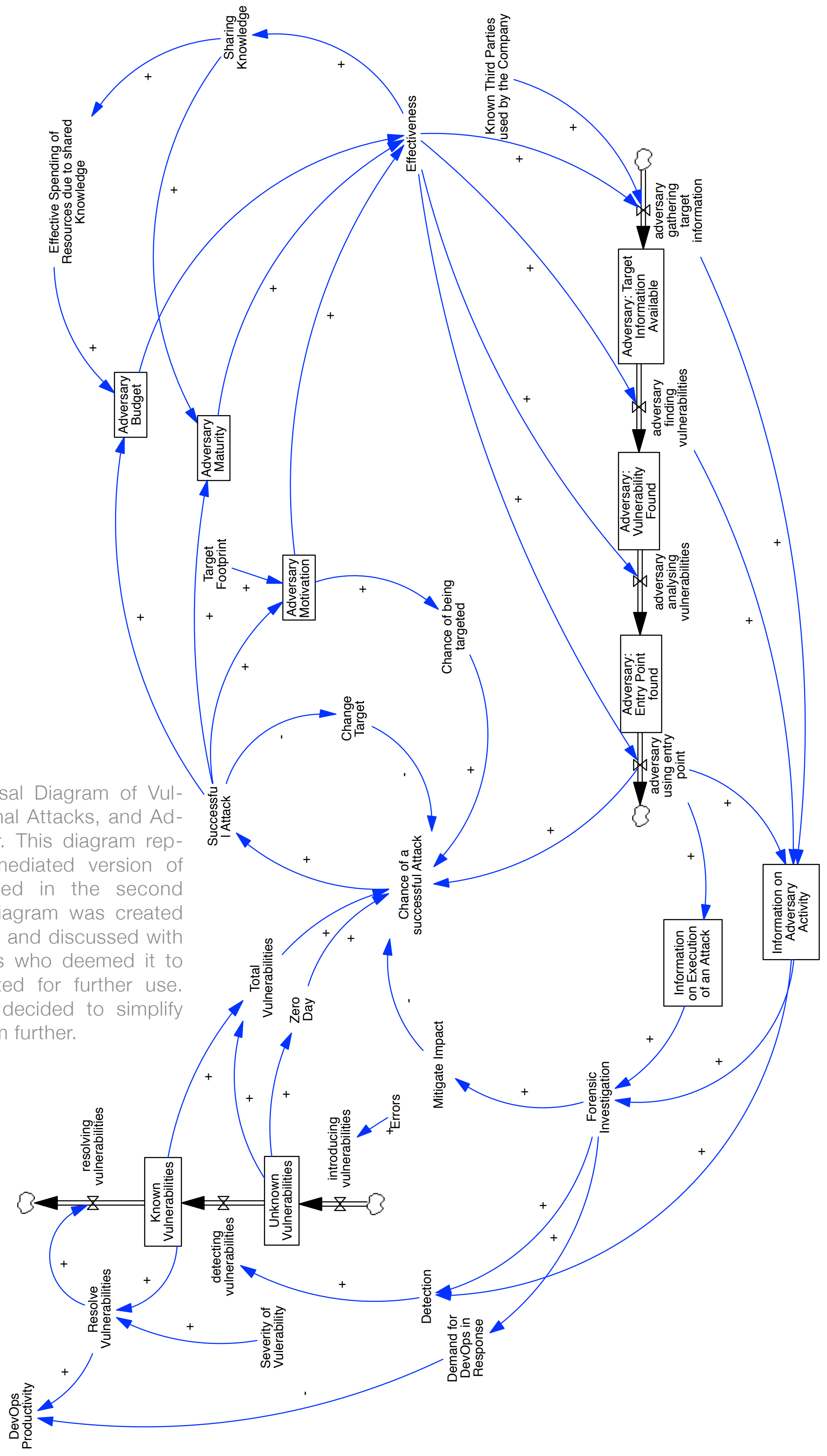

Sharing Knowledge
Effective Spending of Resources due to shared Knowledge
Effectiveness
Known Third Parties used by the Company
adversary gathering target information
Adversary: Target Information Available
adversary finding vulnerabilities
Adversary: Vulnerability Found
adversary analysing vulnerabilities
Adversary: Entry Point found
adversary using entry point
Information on Adversary Activity
Information on Execution of an Attack
Adversary Budget
Adversary Maturity
Target Footprint
Adversary Motivation
Chance of being targeted
Change Target
Successfu l Attack
Chance of a successful Attack
Total Vulnerabilities
Zero Day
Mitigate Impact
Forensic Investigation
Errors
introducing vulnerabilities
Unknown Vulnerabilities
Known Vulnerabilities
resolving vulnerabilities
detecting vulnerabilities
Resolve Vulnerabilities
Severity of Vulerability
Detection
Demand for DevOps in Response
DevOps Productivity
+
-

Figure I.B.2: Causal Diagram of Vulnerabilities, External Attacks, and Adversary Behaviour. This diagram represents an intermediated version of the model created in the second workshop. The diagram was created by the researcher, and discussed with a few participants who deemed it to be too complicated for further use. Together, it was decided to simplify the causal diagram further.

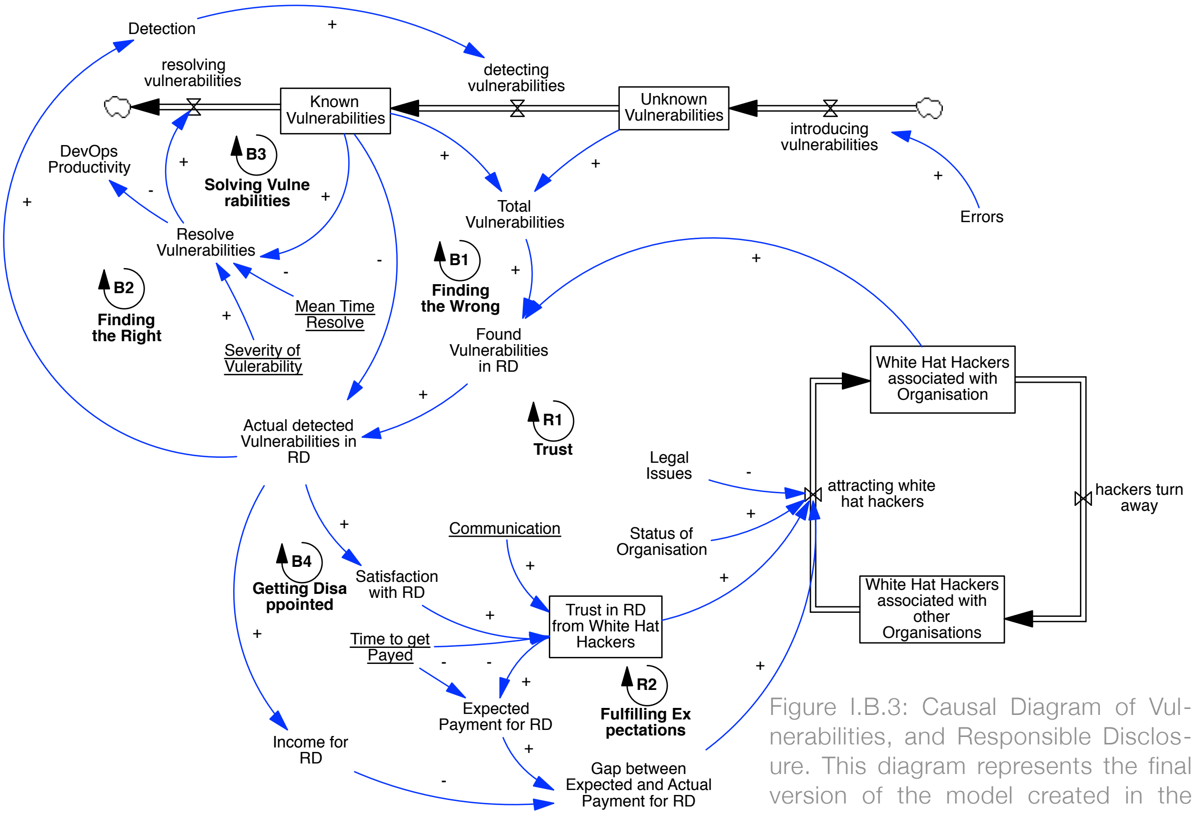


Figure I.B.3: Causal Diagram of Vulnerabilities, and Responsible Disclosure. This diagram represents the final version of the model created in the second workshop. The diagram was created by the researcher and validated by the participants.

As indicated above, each of the points below describes the dynamics of a feedback loop discussed within the session. While the causal diagram in Figure I.B.1 showed the interaction between software vulnerabilities, external cyber attacks, and adversary dynamics, the diagram in Figure I.B.3 depicts the relationship between software vulnerabilities and responsible disclosure. Despite its usefulness for the discussion of practical implications (particularly regarding mean time to resolve), responsible disclosure has been excluded from the actual analysis above because it describes a potential but not necessary part of the investigated interaction.

- B1: Finding unknown, and known but unresolved vulnerabilities.
- B2: Finding only unknown vulnerabilities to pursue the aim of RD.
- B3: Vulnerabilities are resolved.
- B4: Mismatching payment expectations makes hackers turn away.
- R1: Satisfying hackers creates trust and attracts further hackers.
- R2: Matching payment expectations attracts hackers.

- Communication, time to get paid and mean time to resolve are important leverages for attracting hackers, while the severity of a vulnerability is important for deciding on whether to resolve a vulnerability.

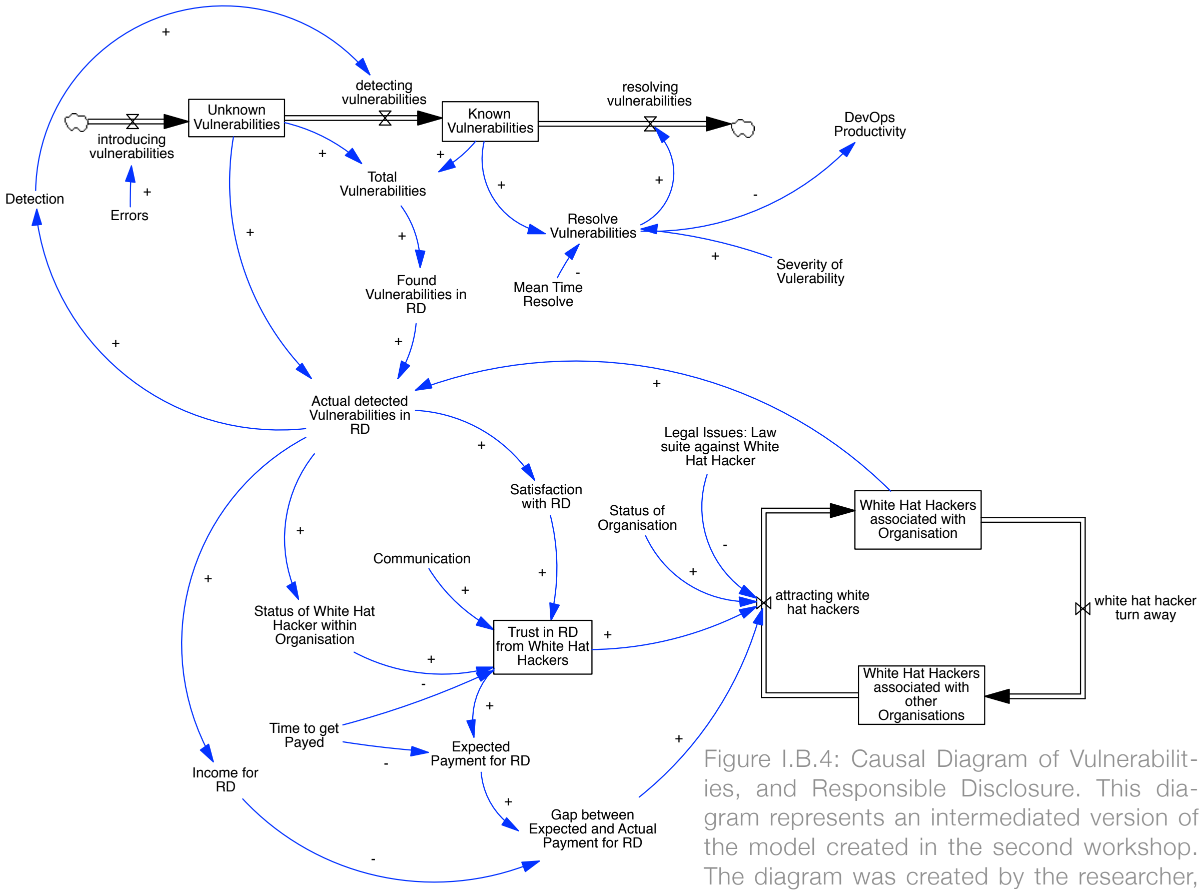

Unknown Vulnerabilities
detecting vulnerabilities
Known Vulnerabilities
resolving vulnerabilities
DevOps Productivity
introducing vulnerabilities
Errors
Detection
Total Vulnerabilities
Resolve Vulnerabilities
Severity of Vulerability
Mean Time Resolve
Found Vulnerabilities in RD
Actual detected Vulnerabilities in RD
Legal Issues: Law suite against White Hat Hacker
Satisfaction with RD
Status of Organisation
White Hat Hackers associated with Organisation
attracting white hat hackers
white hat hacker turn away
White Hat Hackers associated with other Organisations
Communication
Status of White Hat Hacker within Organisation
Trust in RD from White Hat Hackers
Time to get Payed
Expected Payment for RD
Income for RD
Gap between Expected and Actual Payment for RD
+
-

Figure I.B.4: Causal Diagram of Vulnerabilities, and Responsible Disclosure. This diagram represents an intermediated version of the model created in the second workshop. The diagram was created by the researcher, and discussed with a few participants who deemed it to be too complicated for further use. Together, it was decided to simplify the causal diagram further.

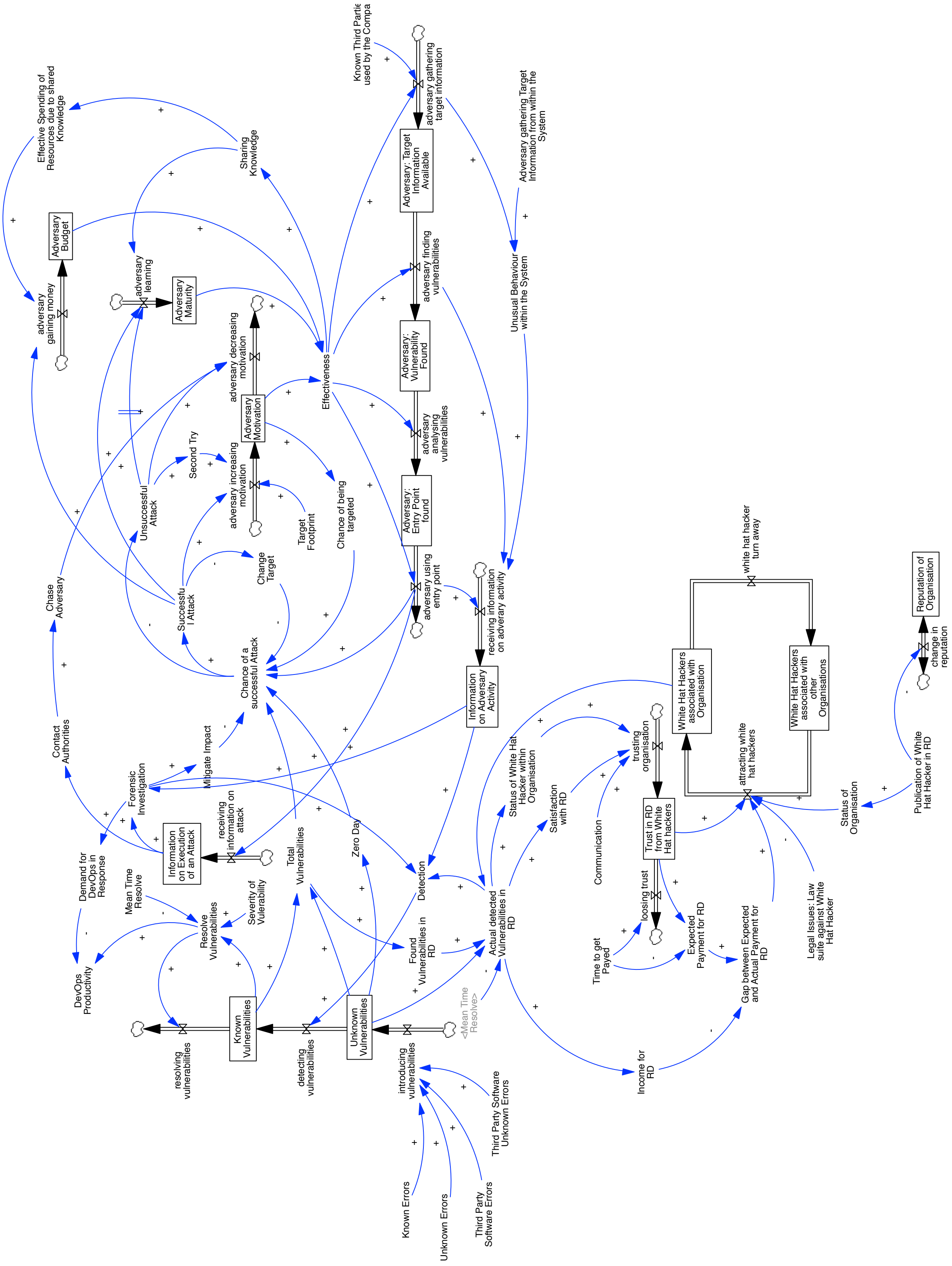

Known Third Partie used by the Compa
adversary gathering target information
Adversary gathering Target Information from within the System
Effective Spending of Resources due to shared Knowledge
Sharing Knowledge
Adversary: Target Information Available
Adversary Budget
adversary gaining money
adversary learning
Adversary Maturity
adversary finding vulnerabilities
Unusual Behaviour within the System
adversary decreasing motivation
Adversary Motivation
Effectiveness
Adversary: Vulnerability Found
adversary analysing vulnerabilities
Second Try
adversary increasing motivation
Unsuccessful Attack
Target Footprint
Chance of being targeted
Adversary: Entry Point found
white hat hacker turn away
Reputation of Organisation
Chase Adversary
Successful Attack
Change Target
adversary using entry point
receiving information on adverary activity
change in reputation
Chance of a successful Attack
Information on Adversary Activity
White Hat Hackers associated with Organisation
White Hat Hackers associated with other Organisations
Contact Authorities
Mitigate Impact
Status of White Hat Hacker within Organisation
trusting organisation
Forensic Investigation
receiving information on attack
Satisfaction with RD
attracting white hat hackers
Status of Organisation
Publication of White Hat Hacker in RD
Information on Execution of an Attack
Zero Day
Total Vulnerabilities
Communication
Trust in RD from White Hat hackers
Demand for DevOps in Response
Mean Time Resolve
Detection
Severity of Vulerability
Resolve Vulnerabilities
losing trust
Legal Issues: Law suite against White Hat Hacker
Actual detected Vulnerabilities in RD
Found Vulnerabilities in RD
Expected Payment for RD
Time to get Payed
Gap between Expected and Actual Payment for RD
DevOps Productivity
<Mean Time Resolve>
Known Vulnerabilities
Unknown Vulnerabilities
resolving vulnerabilities
detecting vulnerabilities
introducing vulnerabilities
Income for RD
Third Party Software Unknown Errors
Known Errors
Unknown Errors
Third Party Software Errors
+
-

Figure I.B.5: Causal Diagram of Software Vulnerabilities, External Cyber Attacks, Adversary Behaviour, and Responsible Disclosure. This diagram represents the initial version of the model created in the second workshop. The diagram was created by the participants who drew it on paper. Afterwards, the researcher translated the diagram onto the computer but kept everything as it was decided by the participants. Once more, to reduce complexity, the researcher decided to simplify this diagram as done with the versions above.

I. C Causal Diagrams Group Model Building Session 3

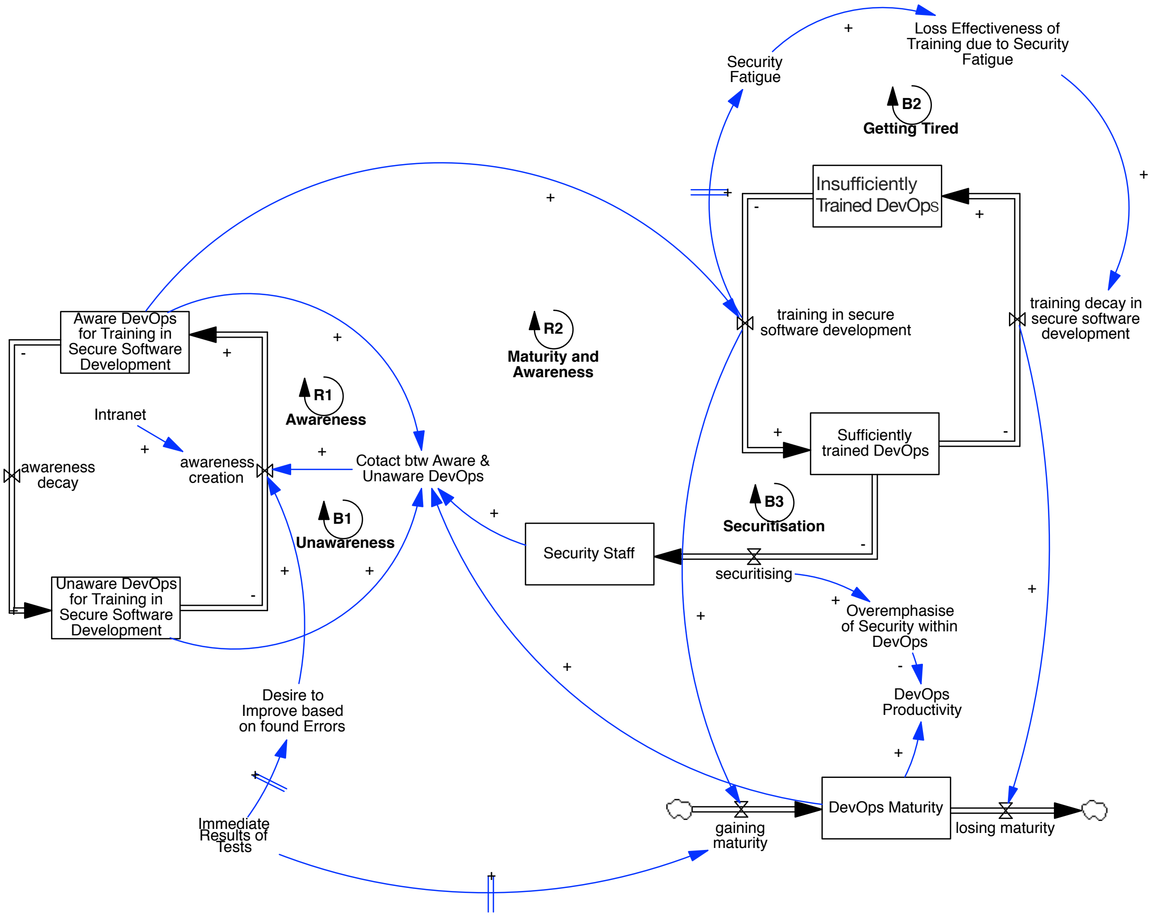

Security Fatigue
Loss Effectiveness of Training due to Security Fatigue
B2
Getting Tired
Insufficiently Trained DevOps
training in secure software development
training decay in secure software development
Sufficiently trained DevOps
B3
Securitisation
securitising
Security Staff
Overemphasise of Security within DevOps
DevOps Productivity
DevOps Maturity
gaining maturity
losing maturity
R2
Maturity and Awareness
Aware DevOps for Training in Secure Software Development
Unaware DevOps for Training in Secure Software Development
awareness decay
Intranet
awareness creation
R1
Awareness
B1
Unawareness
Cotact btw Aware & Unaware DevOps
Desire to Improve based on found Errors
Immediate Results of Tests
+
-

Figure I.C.1: Causal Diagram of DevOps, Maturity, and Awareness. This diagram represents the final version of the model created in the third workshop. In contrast to the previous diagram, this final version resembles very much the model created in the workshop. Minor changes were made by the researcher when translating the model from paper to the computer but those were only about the order and neatness of the model, not about simplifications and parsimony. Although this diagram was finalised after the last session, it was validated by the participants via further communication.

Like with the first and second session, also the results of the third workshop were sent to the participants as a summary shortly after the meeting. Again, instead of workbooks a short and comprehensive summary was created in powerpoint. The presentation simply depicted Figure I.C.1 above and I.C.2 below as final summarised version of the causal diagrams of the second workshop. Of course, the explanatory bullet point list below as well as the one following I.C.2 were added to the powerpoint presentation. Each point describes the dynamics of a feedback loop discussed within the session.

- B1: Unaware DevOps become aware due to aware DevOps.
- B2: Too much Training causes Security Fatigue, decreasing the effect of Training.
- B3: Overtraining of DevOps leads to more Security Staff and less DevOps.

- R1: The more aware DevOps, the less unaware DevOps.
- R2: The more Maturity, the more Awareness. In turn, more Maturity due to more Training.

Particularly this time, the participants were invited to comment on the summary via e-mail, phone call, or in person as there was no further session. Despite the invitation to adjust the diagrams, the participants had not objected to it, but instead, had been very satisfied with the

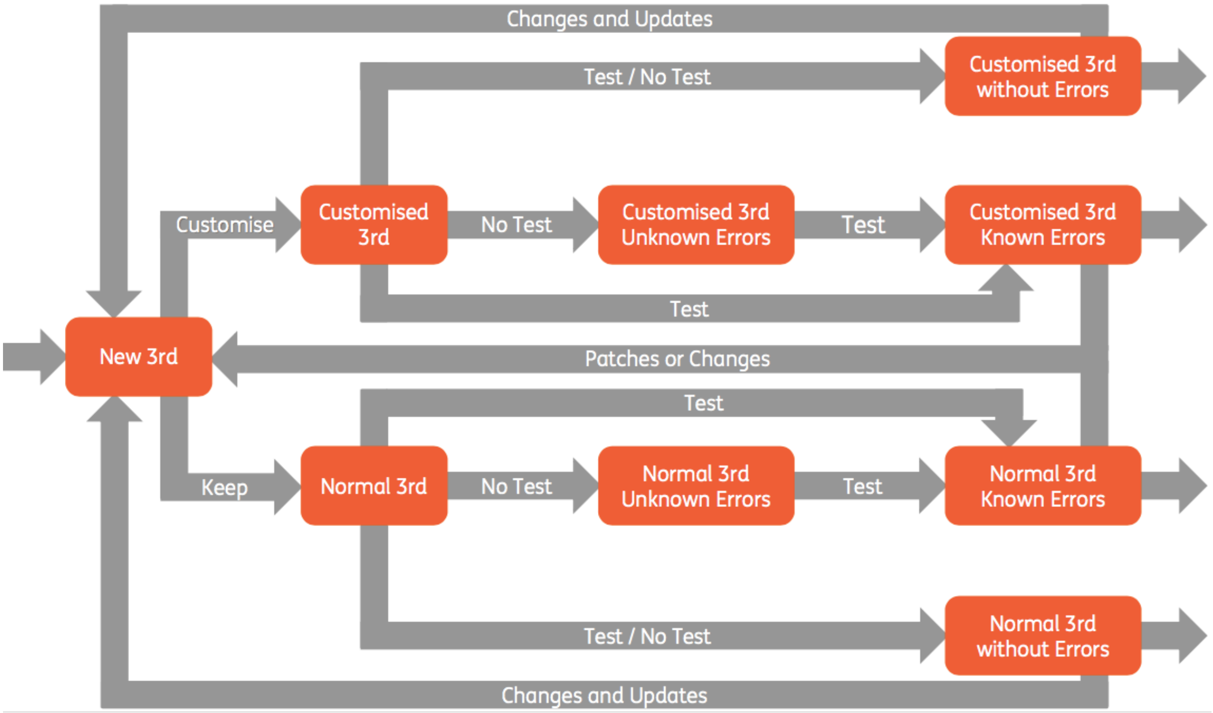


Figure I.C.2: Causal Diagram of Third Party Software. This diagram represents the final version of the model created in the third workshop. During the workshop it turned out that Third Party Software in the setting of this study is less an issue of feedback loops but more one of options in decision making. Hence, the researcher translated the paper model into this diagram to show the different options when dealing with Third Party Software. Despite its different appearance, the diagram depicted here is quite similar to the one created during the group model building workshop. Hence, this final version resembles very much the model created in the workshop. Although this diagram was finalised after the last session, it was validated by the participants via further communication methods.

In contrast to the previous explanations, here the bullet points explain noteworthy points about third party software because there was no focus on the feedback structure. While the causal diagram in Figure I.C.1 showed the interaction between DevOps, awareness, and maturity, this diagram treats third party software. Since it was indicated by the participants during the session that this topic has little relevance for the issue under investigation it was only addressed when necessary but elsewise left out.

- Third party software describes any external software used in the organisation.

- Third party software is generally introduced when new solutions are needed or former software was decommissioned.
- DevOps may work on third party software in the form of...
    - ... searching and deciding on new options,
    - ... customising,
    - ... testing,
    - ... analysing in case of problems and errors,
    - ... communicating with contract partner.
- Work on third party software has generally no impact on DevOps Productivity since it is accounted for in the sprint, except it is conducted excessively.
- Errors in third party software cause Vulnerabilities.

I. D Overarching Causal Diagram Group Model Building for Session 3

Notwithstanding the fact, that the causal diagrams depicted in the Figures I.C.1 and I.C.2 had to be developed with the participants at the beginning of the third workshop, the researcher had created an overarching model including all submodels covered throughout the workshops (see also the script of the first session in which the submodels are presented (II.A.1)) prior to the third session (Figure I.D.1). This model served as a basis to conduct a "model walkthrough" to discuss the connections between the different submodels and to investigate the overarching dynamics with the participants in the third session. Similar to the beginning of the second and third session, the researcher presented the model by unfolding it loop per loop and requested the participants to criticise, improve and adjust the causal diagram according to their own ideas. In other words, the researcher continuously probed the participants to disconfirm the causal diagram in order to increase its internal validity (Andersen et al., 2012; Yin, 2014). Despite the invitation to adjust the causal diagrams, the participants did not object to it, but instead, were very satisfied with the outcomes. Of course, the diagram could not show the insights gained within the third session as it was developed by the researcher prior to the workshop. The insights from the third workshop were included into the validated diagram after the third session.

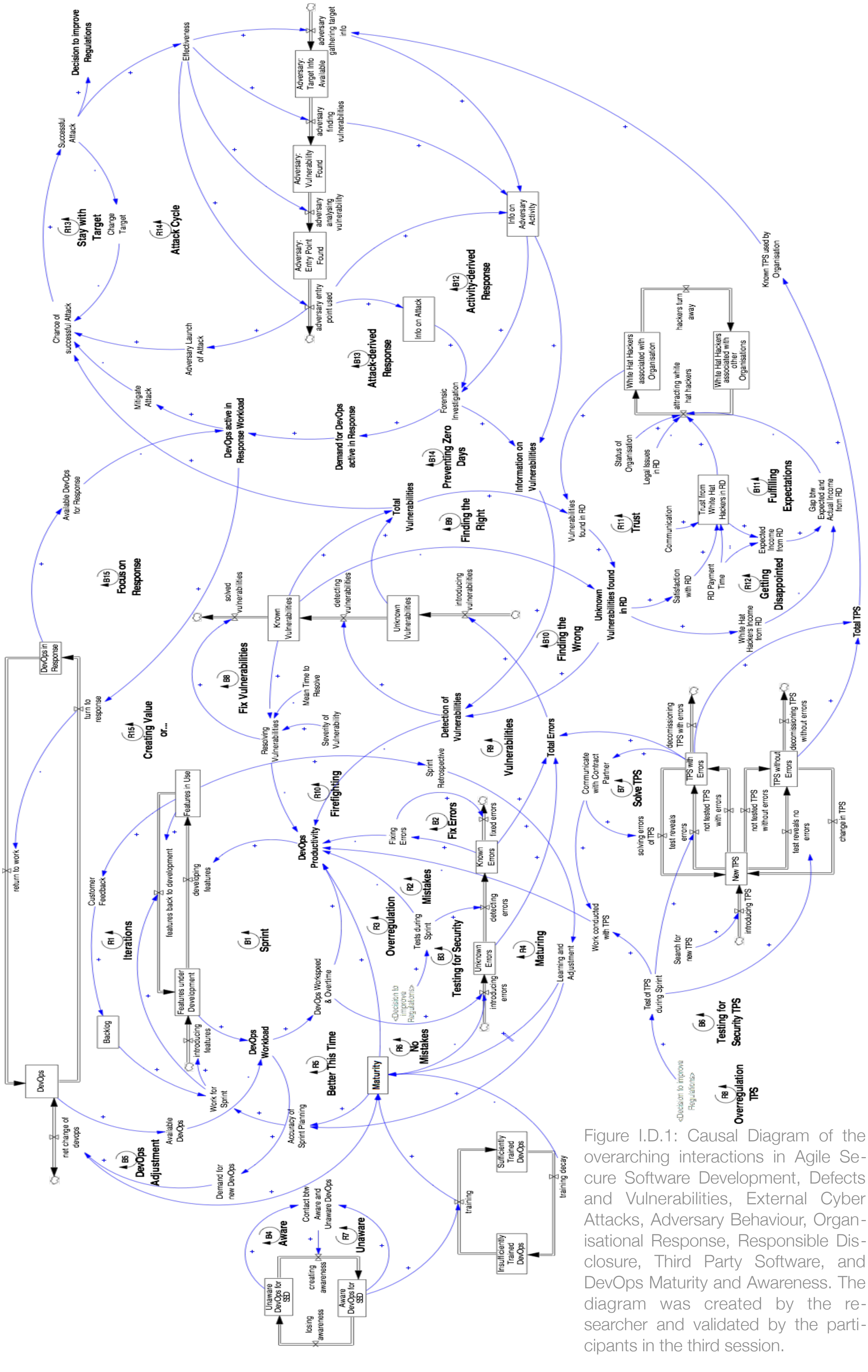

Decision to improve Regulations
Successful Attack
R13 Stay with Target
Change Target
R14 Attack Cycle
Chance of successful Attack
Effectiveness
Adversary: Target Info Available
adversary gathering target info
adversary finding vulnerabilities
Adversary: Vulnerability Found
adversary analysing vulnerability
Adversary: Entry Point Found
adversary entry point used
Adversary Launch of Attack
Mitigate Attack
DevOps active in Response Workload
Demand for DevOps active in Response
Info on Attack
B13 Attack-derived Response
Info on Adversary Activity
B12 Activity-derived Response
Forensic Investigation
B14 Preventing Zero Days
Information on Vulnerabilities
Known TPS used by Organisation
White Hat Hackers associated with Organisation
hackers turn away
White Hat Hackers associated with other Organisations
attracting white hat hackers
Status of Organisation
Legal Issues in RD
Trust from White Hat Hackers in RD
B11 Fulfilling Expectations
Gap btw Expected and Actual Income from RD
Expected Income from RD
Communication
R11 Trust
Vulnerabilities found in RD
Unknown Vulnerabilities found in RD
Satisfaction with RD
RD Payment Time
R12 Getting Disappointed
White Hat Hackers Income from RD
Total TPS
Available DevOps for Response
B15 Focus on Response
solved vulnerabilities
Known Vulnerabilities
detecting vulnerabilities
Unknown Vulnerabilities
introducing vulnerabilities
Total Vulnerabilities
B9 Finding the Right
B10 Finding the Wrong
B8 Fix Vulnerabilities
Mean Time to Resolve
Resolving Vulnerabilities
Severity of Vulnerability
Detection of Vulnerabilities
R9 Vulnerabilities
Total Errors
DevOps in Response
turn to response
return to work
R15 Creating Value or...
Features in Use
Sprint Retrospective
R10 Firefighting
Communicate with Contract Partner
B7 Solve TPS
decomissioning TPS with errors
TPS with Errors
decomissioning TPS without errors
TPS without Errors
solving errors of TPS
test reveals errors
not tested TPS with errors
not tested TPS without errors
test reveals no errors
change in TPS
New TPS
introducing TPS
Search for new TPS
Customer Feedback
features back to development
developing features
DevOps Productivity
Fixing Errors
B2 Fix Errors
fixed errors
Known Errors
R2 Mistakes
detecting errors
Work conducted with TPS
R1 Iterations
B1 Sprint
R3 Overregulation
Tests during Sprint
B3 Testing for Security
Unknown Errors
R4 Maturing
Learning and Adjustment
Test of TPS during Sprint
B6 Testing for Security TPS
Backlog
Features under Development
introducing features
DevOps Workspeed & Overtime
<Decision to improve Regulations>
introducing errors
R6 No Mistakes
DevOps
DevOps Workload
Work for Sprint
R5 Better This Time
Maturity
<Decision to improve Regulations>
R8 Overregulation TPS
net change of devops
Available DevOps
B5 DevOps Adjustment
Accuracy of Sprint Planning
Demand for new DevOps
Sufficiently Trained DevOps
training decay
training
Insufficiently Trained DevOps
Contact btw Aware and Unaware DevOps
B4 Aware
R7 Unaware
creating awareness
Unaware DevOps for SSD
Aware DevOps for SSD
losing awareness

Figure I.D.1: Causal Diagram of the overarching interactions in Agile Secure Software Development, Defects and Vulnerabilities, External Cyber Attacks, Adversary Behaviour, Organisational Response, Responsible Disclosure, Third Party Software, and DevOps Maturity and Awareness. The diagram was created by the researcher and validated by the participants in the third session.

I. E Overarching Causal Diagram Group Model Building after Session 3

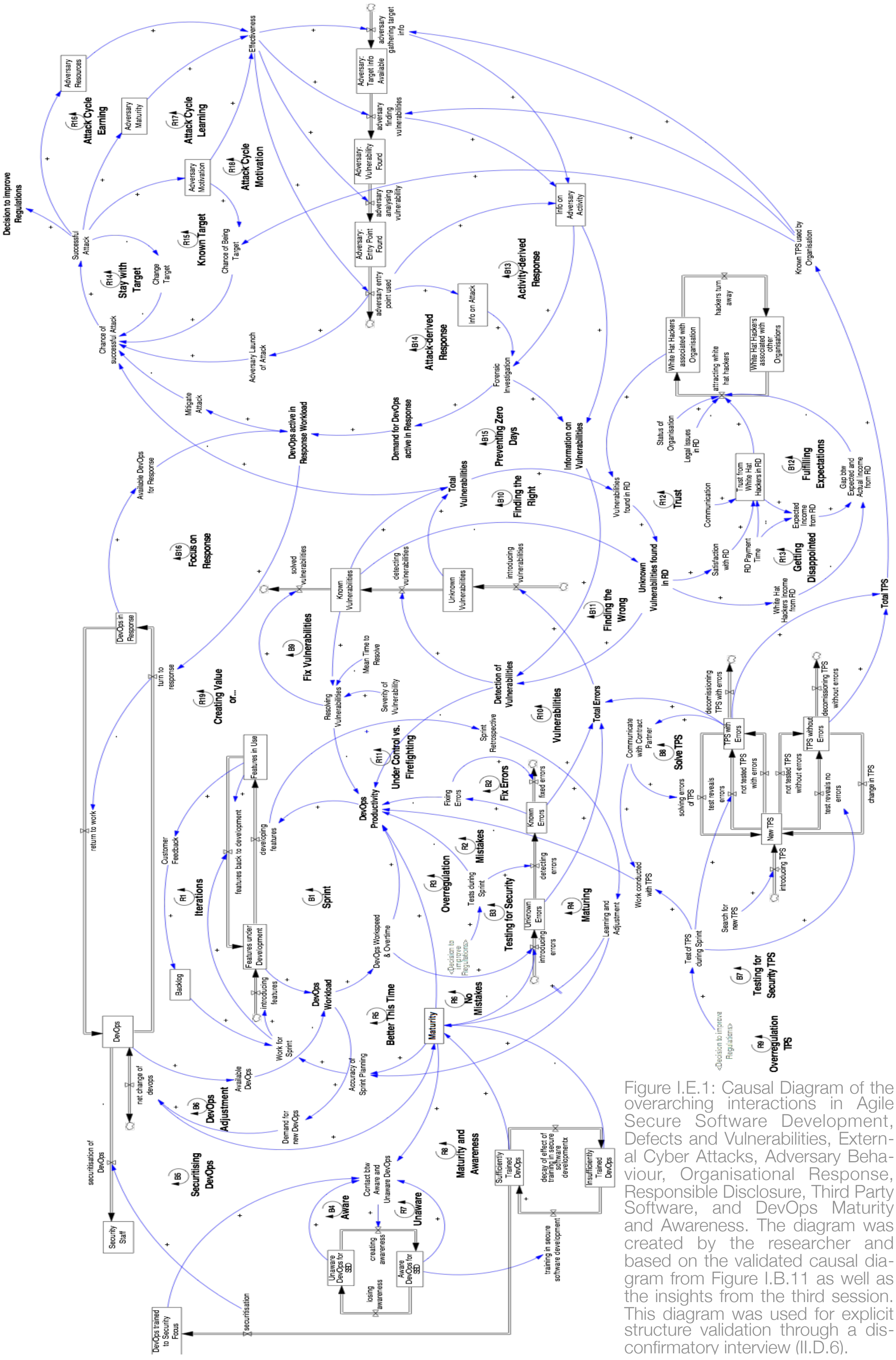

Decision to improve Regulations
Adversary Resources
R16 Attack Cycle Earning
Adversary Maturity
R17 Attack Cycle Learning
Effectiveness
Adversary Motivation
R18 Attack Cycle Motivation
Successful Attack
R14 Stay with Target
Change Target
R15 Known Target
Chance of Being Target
Chance of successful Attack
Adversary Launch of Attack
Mitigate Attack
adversary gathering target info
Adversary: Target info Available
adversary finding vulnerabilities
Adversary: Vulnerability Found
adversary analysing vulnerability
Adversary: Entry Point Found
adversary entry point used
Info on Adversary Activity
B13 Activity-derived Response
Info on Attack
B14 Attack-derived Response
Forensic Investigation
Demand for DevOps active in Response
DevOps active in Response Workload
Available DevOps for Response
B15 Preventing Zero Days
Information on Vulnerabilities
Known TPS used by Organisation
White Hat Hackers associated with Organisation
hackers turn away
White Hat Hackers associated with other Organisations
attracting white hat hackers
Status of Organisation
Legal issues in RD
Trust from White Hat Hackers in RD
B12 Fulfilling Expectations
Gap btw Expected and Actual Income from RD
Communication
R12 Trust
Expected Income from RD
Vulnerabilities found in RD
Total Vulnerabilities
B10 Finding the Right
B16 Focus on Response
solved vulnerabilities
Known Vulnerabilities
detecting vulnerabilities
Unknown Vulnerabilities
introducing vulnerabilities
Unknown Vulnerabilities found in RD
Satisfaction with RD
RD Payment Time
R13 Getting Disappointed
White Hat Hackers income from RD
Total TPS
B11 Finding the Wrong
B9 Fix Vulnerabilities
Mean Time to Resolve
Severity of Vulnerability
Resolving Vulnerabilities
Detection of Vulnerabilities
R10 Vulnerabilities
Total Errors
DevOps in Response
turn to response
return to work
R19 Creating Value or...
Features in Use
Customer Feedback
features back to development
developing features
R1 Iterations
Features under Development
introducing features
Backlog
B1 Sprint
DevOps Productivity
R11 Under Control vs. Firefighting
Sprint Retrospective
Fixing Errors
B2 Fix Errors
fixed errors
Known Errors
R2 Mistakes
R3 Overregulation
Tests during Sprint
B3 Testing for Security
detecting errors
Unknown Errors
introducing errors
R4 Maturing
Learning and Adjustment
Work conducted with TPS
Communicate with Contract Partner
B8 Solve TPS
solving errors of TPS
test reveals errors
TPS with Errors
decommissioning TPS with errors
not tested TPS with errors
New TPS
introducing TPS
not tested TPS without errors
TPS without Errors
decommissioning TPS without errors
test reveals no errors
change in TPS
Search for new TPS
Test of TPS during Sprint
B7 Testing for Security TPS
&lt;Decision to improve Regulations&gt;
R9 Overregulation TPS
DevOps Workspeed &amp; Overtime
&lt;Decision to improve Regulations&gt;
R6 No Mistakes
DevOps
DevOps Workload
Work for Sprint
R5 Better This Time
Maturity
Accuracy of Sprint Planning
Available DevOps
net change of devops
B6 DevOps Adjustment
Demand for new DevOps
R8 Maturity and Awareness
securitisation of DevOps
B5 Securitising DevOps
Security Staff
Contact btw Aware and Unaware DevOps
B4 Aware
R7 Unaware
Unaware DevOps for SD
creating awareness
Aware DevOps for SD
losing awareness
Sufficiently Trained DevOps
decay of effect of training in secure software development
Insufficiently Trained DevOps
training in secure software development
DevOps trained to Security Focus
securitisation

Figure I.E.1: Causal Diagram of the overarching interactions in Agile Secure Software Development, Defects and Vulnerabilities, External Cyber Attacks, Adversary Behaviour, Organisational Response, Responsible Disclosure, Third Party Software, and DevOps Maturity and Awareness. The diagram was created by the researcher and based on the validated causal diagram from Figure I.B.11 as well as the insights from the third session. This diagram was used for explicit structure validation through a disconfirmatory interview (II.D.6).

I. F Overarching Causal Diagram Group Model Building after Validation

After the disconfirmatory interview with a system architect and security expert (II.D.6) a final version of the causal diagram evolved. This model is depicted in Figure I.F.1. Like with all the previous results from the workshops, this causal diagram was sent to the participants and all others involved in the study as a summary shortly after the validation session with the system architect. As previously done as well, all feedback loops or other important aspects were explained through brief comments in the powerpoint presentation which served as the summary. Since the presentation unfolded the overarching model step by step, the summary contained more than forty pages. The feedback from the participants and the involved team from the security department was very good and no changes were requested. Instead, the insights from the study were confirmed, for instance, in a later team meeting in which the researcher presented the summary to the team (II.D.17). Below, all comments in the presentation are listed.

- B1: Developing Software within Sprint.
- R1: Based on Feedback new Items are added to the Backlog.
- B2, R2: Longer work hours and less time per item increase the productivity but also the number of errors which need to be fixed.
- B3, R3: Tests reveal unknown errors. However, under extreme conditions, tests would decrease productivity if conducted excessively.
- R4, R5, R6: Retrospective improves the next sprint. Additionally, over time maturity increases, resulting in fewer errors.
- B6: Next to automated solutions, time per task and overtime, the number of DevOps determines the capacity to develop software.

  Insight: Hence, the number of DevOps is critical for the workload of a team.
- B4, R7, R8: Training increases maturity. To take part in trainings, DevOps need to know that training possibilities are available. Training, however, also decreases the productivity of a team while they are in training (not shown in the diagram).
- B5: While maturity is important for software quality, mature staff is also more likely to change job.
- Next to internally developed software, third party software is in use. Such software may be tested and can have flaws.
- B7, B8, R9: Testing increases security. Flaws in software cause work and decrease productivity.

- Known and unknown errors of internal and third party software cause vulnerabilities which may be exploited.
- B9, B10: Critical vulnerabilities have to be fixed without undue delay. Other vulnerabilities are put on the backlog and fixed later.
  Insight: While fixing less critical vulnerabilities later decreases disruptions to the sprint backlog, it causes strategic delays in the long term.
- R10a, R10b: Vulnerabilities cause future work that becomes known with a delay.
  Insight: Hence, not accounting for the future work of fixing vulnerabilities means adapting to a wrong future workload.
- R16, R17, R18: Vulnerabilities enable successful attacks which in turn increase the motivation, maturity and resources of hackers.
- Hackers search for information and vulnerabilities to execute an attack.
- B14, R14: Hackers may change target to exploit the same vulnerability elsewhere or if unsuccessful. Hackers may stay with the same target to use the gained insights or if that simply is part of the overall strategy.
- R15: The more known a target, the more likely an attack. Other attack vectors may serve as entry point.
- B17: Adversary activities leave their marks, and help to detect unknown vulnerabilities.
- B14: Additionally, if detected adversary activities cause a response to mitigate the attack or minimise the impact of it.
- B15: Further, if detected executed attacks cause a response as well. Together, these information help to detect vulnerabilities and prevent zero days.
- B16: A common way to security is to first adjust the firewall and later resolve the vulnerability. This is, however, not in all cases possible, thus the strength of this mechanism is ambiguous.
- Next to best practice throughout the sector, successful attacks lead to new regulations (e.g. tests) to improve security.
- Using third party software provides an adversary with information and may increase the likelihood of being a target.
- B11, B12: Within responsible disclosure, unknown vulnerabilities are to be found by external ethical hackers.
- R12: Ethical hackers have the option to collaborate with the organisation or with other companies. Trust due to past experience is what binds them to the organisation.

- B13, R13: Additionally, matching their payment expectations, communicating with them and solving vulnerabilities keeps them collaborative.
- R11: On the one hand, little errors from the beginning cause little work later and keep the system in balance. On the other hand, many errors and vulnerabilities make teams constantly lagging behind.

  Insight: This development may evolve in two directions: Either a virtuous circle of less problems, or a vicious circle of continuous firefighting.
- B18, R19: Finally, for mitigating an attack, DevOps in response are needed. After having mitigated, DevOps turn back to their normal work.

  Insight: Again, this may lead to a development evolving in two directions: Either a virtuous circle of little responses and value creation, or a vicious circle of continuous mitigation and the erosion of business.
- Critical feedback interrelations of internal and external processes:
  - B6
  - B10
  - R10a, R10b
  - R11
  - R19
- Possible Way to Measure Improvement proposed by the participants and summarised and refined by the researcher:
  - Maturity and Training.
  - DevOps Workload and DevOps Productivity.
  - Errors per Feature, Test Results, and “Technical Debt”.
  - Detected Vulnerabilities or Known Vulnerabilities.
  - Mean Time to Resolve Vulnerabilities.
  - Attempted Attacks and Successful Attacks.
  - Responsible Disclosure Reports of Unknown Vulnerabilities.
- Possible areas of improvement proposed by the participants and summarised and refined by the researcher:
  - Increase maturity (training, experience, dedication).
  - Train staff and create culture of dedication.
  - Keep mature staff.
  - “Lead by example” and “train the trainers”.
  - Create awareness culture.
  - “Share success but also mistakes”.

- Follow Agile approach and collaborate with customer.
- Test early and test for defects with automated solutions.
- Improve capability to detect vulnerabilities, increase the information on vulnerabilities, find zero days before adversary, and fix fast (decrease mean time to resolve).
- Fast feedback for DevOps to have knowledge on quality without delay.
- Keeping productivity high while avoiding firefighting vulnerabilities.
- Get more white hat hackers (ethical hackers) associated with the organisation to improve responsible disclosure.
- Specialised emergency teams.
- Display attacks in real time.

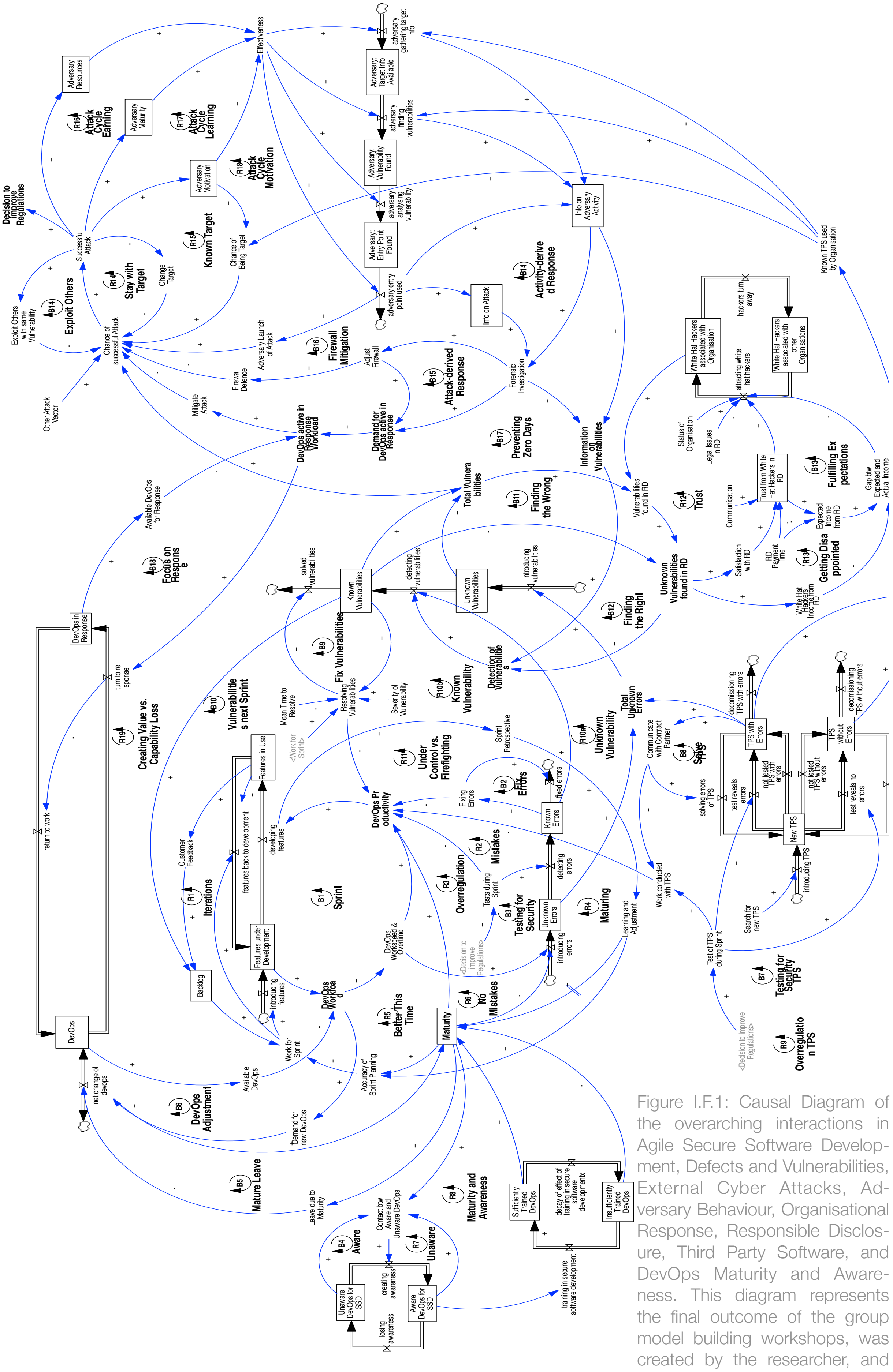

Adversary Resources
Adversary Maturity
Adversary Motivation
Effectiveness
adversary gathering target info
Adversary: Target Info Available
adversary finding vulnerabilities
Adversary: Vulnerability Found
adversary analysing vulnerability
Adversary: Entry Point Found
adversary entry point used
Info on Attack
Info on Adversary Activity
Known TPS used by Organisation
R16 Attack Cycle Earning
R17 Attack Cycle Learning
R18 Attack Cycle Motivation
Decision to improve Regulations
Successful Attack
R15 Known Target
Chance of Being Target
R14 Stay with Target
Change Target
Exploit Others with same Vulnerability
B14 Exploit Others
Chance of successful Attack
Other Attack Vector
Adversary Launch of Attack
B16 Firewall Mitigation
Adjust Firewall
Firewall Defence
Mitigate Attack
B15 Attack-derived Response
Forensic Investigation
B14 Activity-derived Response
DevOps active in Response Workload
Demand for DevOps active in Response
B17 Preventing Zero Days
Information on Vulnerabilities
White Hat Hackers associated with Organisation
hackers turn away
White Hat Hackers associated with other Organisations
attracting white hat hackers
Status of Organisation
Legal Issues in RD
Trust from White Hat Hackers in RD
B13 Fulfilling Expectations
Gap btw Expected and Actual Income
Communication
R12 Trust
Vulnerabilities found in RD
Total Vulnerabilities
B11 Finding the Wrong
Available DevOps for Response
B18 Focus on Response
solved vulnerabilities
Known Vulnerabilities
detecting vulnerabilities
Unknown Vulnerabilities
introducing vulnerabilities
Unknown Vulnerabilities found in RD
Satisfaction with RD
RD Payment Time
Expected Income from RD
R13 Getting Disappointed
White Hats Hackers Income from RD
B12 Finding the Right
DevOps in Response
turn to response
R19 Creating Value vs. Capability Loss
B10 Vulnerabilities next Sprint
B9 Fix Vulnerabilities
Mean Time to Resolve
<Work for Sprint>
Resolving Vulnerabilities
Severity of Vulnerability
R10b Known Vulnerability
Detection of Vulnerabilities
Total Unknown Errors
R10a Unknown Vulnerability
Features in Use
Sprint Retrospective
R11 Under Control vs. Firefighting
Communicate with Contract Partner
B8 Spee
decommissioning TPS with errors
decommissioning TPS without errors
TPS with Errors
TPS without Errors
solving errors of TPS
test reveals errors
not tested TPS with errors
not tested TPS without errors
test reveals no errors
New TPS
introducing TPS
Customer Feedback
features back to development
developing features
DevOps Productivity
Fixing Errors
B2 Fix Errors
fixed errors
Known Errors
R1 Iterations
B1 Sprint
R3 Overregulation
R2 Mistakes
Tests during Sprint
detecting errors
B3 Testing for Security
Unknown Errors
R4 Maturing
Learning and Adjustment
Work conducted with TPS
Search for new TPS
return to work
Backlog
Features under Development
introducing features
DevOps Workload
DevOps Workspeed & Overtime
<Decision to improve Regulations>
introducing errors
R6 No Mistakes
Test of TPS during Sprint
B7 Testing for Security TPS
R5 Better This Time
Maturity
DevOps
Work for Sprint
Available DevOps
Accuracy of Sprint Planning
net change of devops
B6 DevOps Adjustment
Demand for new DevOps
<Decision to improve Regulations>
R9 Overregulation TPS
B5 Mature Leave
Leave due to Maturity
R8 Maturity and Awareness
Sufficiently Trained DevOps
decay of effect of training in secure software development
Insufficiently Trained DevOps
B4 Aware
Contact btw Aware and Unaware DevOps
R7 Unaware
Unaware DevOps for SSD
creating awareness
Aware DevOps for SSD
losing awareness
training in secure software development

Figure I.F.1: Causal Diagram of the overarching interactions in Agile Secure Software Development, Defects and Vulnerabilities, External Cyber Attacks, Adversary Behaviour, Organisational Response, Responsible Disclosure, Third Party Software, and DevOps Maturity and Awareness. This diagram represents the final outcome of the group model building workshops, was created by the researcher, and has been validated since then several times.

APPENDIX II - QUALITATIVE RESEARCH

II. A Preparatory Scripts Group Model Building

Throughout the last decades, researcher and practitioners have developed, applied, and assessed several approaches of participatory system dynamics modelling, such as the previously mentioned group model building (Andersen, & Richardson, 1997; Luna-Reyes et al., 2006; Vennix, 1996), and other approaches like mediated modelling (van den Belt, 2004), or participatory system dynamics modelling (Videira, Antunes, & Santons, 2017). While the approaches differ in some details, all of them have relied on the use of so called “scripts” which were first introduced by Andersen and Richardson (1997). Scripts describe individual activities before, within, or after participatory modelling workshops (Andersen, & Richardson, 1997; Luna-Reyes et al., 2006). Combining scripts, other practical guidelines, such as Vennix’ seminal work on group model building (1996), and own experience enables a facilitator (or a facilitation team) to plan, organise, prepare, conduct, and follow-up (a) workshop(s). It is common practice that a facilitator (or a team) creates own “session-scripts” for each individual workshop to improve the modelling process and the outcomes. In other words, such clear, written planning serves as a guideline when preparing for the sessions and conducting the workshop. Relying on best practice and a broad range of scripts when developing session-scripts helps to reap the benefits of established methods and avoid common pitfalls.

Additionally, written session-scripts serve as part of the documentation of the research conducted during the case study, and thereby, increase the reliability of the outcomes (Yin, 2014). Since intervention methods, such as group model building, “have been criticised because it was unclear whether the facilitator or the method contributed most to the results” (Rouwette, & Franco, Unpublished, p. 47), clearly documenting the steps taken in a group model building workshop help to address this unclarity as this makes the results more independent of the researcher. At the same time, it is noteworthy that every workshop is unique and potentially not entirely reproducible because of the social dynamics within groups and the necessary impact of a facilitator on the group (Vennix, 1996). Yet, session-scripts make the procedures transparent, give reason to why certain steps have been taken, enable others to query the outcomes, build on the study, or repeat the research, and finally, minimise errors and biases in the case study.

The following parts below present the used session-scripts for the three group model building workshops conducted within the financial organisation.

II. A. 1 Preparatory Script Group Model Building Session 1

OVERVIEW:

Location: Conference Room within the Financial Organisation[31]

Date: 16.03.2017

Topics: Introduction, Software Development, Errors, DevOps Staff

Facilitator: Jonas Matheus
Recorder:[32] Colleague

Participants: Seven with expertise in Ethical Hacking, Fraud, Penetration Testing, Responsible Disclosure, Software Development, System Architecture, and Vulnerability Scanning

Step	Facilitator	Assistant	Time	Min
Introduction Gatekeeper	-	-	13.00 - 13.05	5
Introduction	Jonas	Colleague	13.05 - 13.10	5
Major Challenges • Nominal Group Technique • Presentation & Comments	Jonas	Colleague	13.10 - 13.20	10
GMB Workshop • Conceptual Overview • System Dynamics	 Jonas Colleague	 Colleague Jonas	13.20 - 13.35 13.20 - 13.25 13.25 - 13.35	15 5 10
Modelling: Software Development, Errors, DevOps I • Software Development • Errors • DevOps (most likely not started) • Third Party Software (most likely not started)	Jonas	Colleague	13.35 - 14.30 13.35 - 14.15 14.15 - 14.30	55 40 15
Break			14.30 - 14.40	10
Modelling: Software Development, Errors, DevOps II • Software Development (most likely finished) • Errors • DevOps (if little time available, if possible both) • Third Party Software (if much time available, if possible both)	Jonas	Colleague	14.40 - 15.45 14.40 - 15.05 15.05 - 15.45 15.05 - 15.45	65 25 40
Conclusion	Jonas	Colleague	15.45 - 16.00	15

[31] The conference room is not indicated in further detail due to confidentiality. In the original session-script the room was given.

[32] Research has indicated that there are up to five roles in a team which conducts group model building workshops, namely the facilitator, the modeller/recorder, content coach, process coach, and the gatekeeper (Andersen, & Richardson, 1997; Vennix, 1996). According to Rouwette, & Franco (Unpublished), the facilitator who guides the workshop and the modeller/recorder who takes note about the session represent the two most important roles in group model building.

0. PREPARATION:

(420 min)
09.00 - 11.00 & 12.00 - 13.00 (180 min)

	Description
Roles	Facilitator: Jonas Assistant: Colleague
Aim	• Preparation of the session • Clarification among the facilitators • Preparation of the location
Steps	0.1 Script preparation 0.2 Preliminary model preparation in Vensim 0.3 Powerpoint presentation 0.4 Preliminary model preparation on paper (on site) 0.5 Clarification among facilitators 0.6 Preparation of location (on site)
Scripts	Scheduling the day, Logistics and room set up, Vennix, 1996

0.1 Script Preparation:

Time available for Task:	180 min
Primary Nature of Task:	Offline
Facilitator:	Jonas

- Look up all scripts on Scriptapedia and check for applicability
- Based on previous (successful) project, prepare new session-scripts

0.2 Preliminary Model in Vensim:

Time available for Task:	120 min
Primary Nature of Task:	Offline
Facilitator:	Jonas

- Focus on small models (Richardson, 2013; Vennix, 1996)
- Decide on “Concept Model” (quantified) or “Preliminary Model” (qualitative)
- Focus on software development, errors, DevOps staff, and third party software

0.3 Powerpoint Presentation:

Time available for Task:	60 min
Primary Nature of Task:	Offline
Facilitator:	Jonas

- Powerpoint presentation for guiding the participants through the workshop and presenting the conceptual model, system dynamics, and the preliminary models
- Based on previous (successful) project, prepare new presentation

0.4 Preliminary Model on Paper:

Time available for Task:	120 min (on site)
Primary Nature of Task:	Offline
Facilitator:	Jonas

- Draw the Vensim models on paper in the facilities of the organisation

0.5 Clarification and Planning among Facilitator and Recorder:

Time available for Task:	60 min
Primary Nature of Task:	Offline
Facilitator:	Jonas & Colleague

- Principal facilitator presents script to the assisting facilitator / recorder
- Discussion on unclear issues or aspects to be changed / improved

0.6 Preparation of Location:

Time available for Task:	60 min (on site)
Primary Nature of Task:	Offline
Facilitator:	Jonas & Colleague

Entirely based on the script on Logistics and Room Set Up

- “Arrange the table, chairs, and flip charts in the room in a manner conducive to upcoming activities and scripts. Let the participants sit in a semicircle facing either the wall where a model is projected, the white board, or the chalkboard.”
    - Around a big table, facing the screen, white board and windows
- “Arrange power cords, tables, and chairs for members not sitting at the table with participants (e.g., recorders, modellers, coaches). Secure any power cords and extension cables with tape to minimise the risk that people may trip.”
    - Power cords in the table, chairs available
- Arrange refreshments in a place that is convenient for participants to get up and access during the session.
    - Refreshments and snacks are provided by the organisation

1. INTRODUCTION:

13.00 - 13.10 (10 min)

	Description
Roles	Facilitator: Jonas Assistant: Colleague
Aim	Introduction of the Session
Steps	1.1 Gatekeeper Introduction 1.2 Facilitator Team Introduction
Scripts	Creating a shared vision of a modelling project (description elements used)

1.1 Gatekeeper Introduction:

Time available for Task:	13.00 - 13.05 (5 min)
Primary Nature of Task:	Presentation

1.2 Facilitator Team Introduction:

Time available for Task:	13.05 - 13.10 (5 min)
Primary Nature of Task:	Presentation & Conversation
Facilitator:	Jonas

- Presentation of Team
    - Names
    - EMSD Programme Radboud University
    - Master thesis in the Organisation
        - Malware & Employees
        - Software Quality & Possibilities of Exploitation
    - No technical background

- Presentation of Participants
    - Names
    - Work description

- Presentation of Project
    - Background information: High software quality decreases the chances of malicious exploitation
    - Problem: Learning about and clarifying interrelations in the system with regard to behavioural (people) and organisational (processes) elements
    - Aim: Improve software quality and thereby decrease possibility of exploitation by better knowing the interrelations within the organisation
    - Resources: Three sessions * each three hours

- Agenda Presentation & Parking Lot Explanation

- Questions

2. MAJOR CHALLENGES IN SOFTWARE QUALITY AND EXPLOITATION:

13.10 - 13.20 (10 min)

	Description
Roles	Facilitator: Jonas Assistant: Colleague
Aim	Receiving the unbiased ideas on topics to be covered
Steps	2.1 Explanation & Nominal Group Technique 2.2 Presentation & Comments
Scripts	Nominal Group Technique, Variable Elicitation
Notes	

2.1 Explanation and Nominal Group Technique:

Time available for Task:	13.10 - 13.15 (5 min)
Primary Nature of Task:	Divergent
Facilitator:	Jonas

- *In order to serve the organisation, we want to make sure that we do not miss something important when covering this topic. Hence, we would like to ask you to write down (in two to three words) one or two major challenges in software quality and possibilities of exploitation. As mentioned at the beginning, we are mainly interested in issues that relate to people and processes, thus, it would be nice to focus your ideas on concerns into this direction. We do this step before we start the actual model building session because we want you to be as little influenced by us as possible.*[33]
- Assistant distributes post its and markers while the facilitator introduces the task.

2.2 Presentation and Comments:

Time available for Task:	13.15 - 13.20 (5 min)
Primary Nature of Task:	Convergent
Facilitator:	Jonas

- Facilitator asks the participants to present one of their ideas within 20 to 30 seconds in a round-robin fashion.
- If necessary, questions and comments for clarification, not for discussion, are possible.
- Questions

3. GROUP MODEL BUILDING WORKSHOP:

13.20 - 13.35 (15 min)

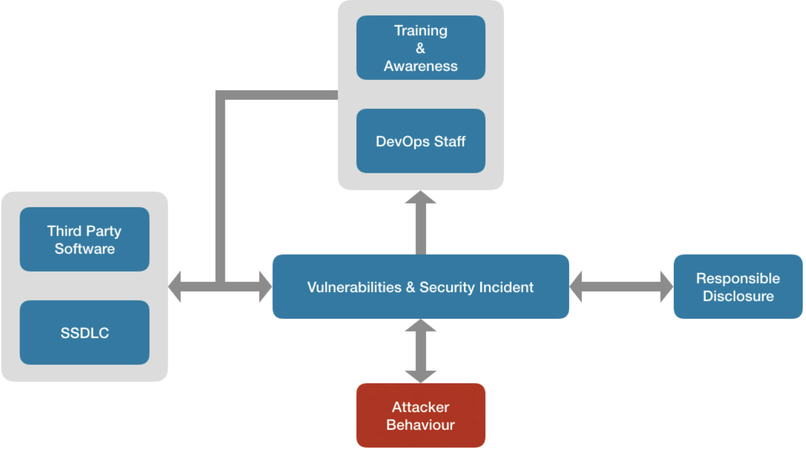
	Description
Roles	Facilitator: Jonas Facilitator: Colleague Assistant: Colleague Assistant: Jonas
Aim	• Explain the planned topics to be covered within this GMB project • Familiarise the participants with the System Dynamics approach
Steps	3.1 Conceptual overview 3.2 System dynamics
Scripts	Vennix, 1996
Notes	

3.1 Conceptual Overview:

Time available for Task:	13.20 - 13.25 (5 min)
Primary Nature of Task:	Presentation
Facilitator:	Jonas

[33] Italics within the session-scripts indicate that the text may be read out loud to the participants.

- Present the conceptual overview of the modelling project, covering all sub-models to be built.
- Emphasise that by definition this is an abstract picture which does not cover reality.
- Instead, the overview is supposed to function as a point of reference.



- Questions

3.2 System Dynamics:

Time available for Task:	13.25 - 13.35 (10 min)
Primary Nature of Task:	Presentation
Facilitator:	Colleague[34]

- Step-by-step presentation of an easy system dynamics model
- Introduction to stocks, flows, converters, links, polarity, delays, feedback loops
- Exercise based on a simple example of the financial sector

- Questions

4. MODELLING - SOFTWARE DEVELOPMENT, ERRORS, AND DEVOPS I:

13.35 - 14.30 (55 min)

	Description
Roles	Facilitator: Jonas Assistant: Colleague
Aim	• Present at least three sub-models • Conduct group model building on these three sub-models • Build the qualitative structure of the issues addresses by the sub-models at hand
Steps	4.1 Software Development 4.2 Errors 4.3 DevOps (most likely not started)
Scripts	Causal mapping with seed structure, concept model, ratio exercise, Vennix, 1996
Notes	

[34] While all other topics during the workshops are content and study related, explaining system dynamics is not which is why the assistant / recorder has taken the role of the facilitator for this part.

4.1 Software Development:

Time available for Task:	13.35 - 14.15 (40 min)
Primary Nature of Task:	Divergent
Facilitator:	Jonas

- Present the preliminary sub-model of software development through the power point presentation
    - Naming:
        - Earlier Stages: Code, Build
        - Later Stages: Deploy, Test, Release
    - Work Pressure: Ratio Exercise as Explanation
        - *The Work Pressure is built by Demand in DevOps and Available DevOps*
        - *What would happen if Demand in DevOps goes to zero?*
        - *What would happen if Available DevOps goes to zero?*
- Clarify for questions
- Put the paper-model on the large table between the participants
    - Provide each participant with sticky notes and a marker, so that they have the opportunity to get engaged
- Invite the participants to discuss, change, and adjust the model and delete parts of it

- In case a discussion does not really come up, the following questions may help to get the participants talk about the topic and interact with the model. The underlined questions are most important:

- Overall Model - Disconfirmation
    - What is missing in the picture?
    - What is wrong in the picture?
    - What would you like to change?
- Process of Software Development
    - What constitutes a high software quality?
    - What may happen if software quality is low?
    - What happens before the planning of software?
    - What happens with software in use?
    - Is it accurate to split the tasks up into these two phases?
    - How does planning and development take place?
        - Which general actions are taken in software development?
        - Are there any standards and processes in place?
        - Are there conditions under which people do not follow standards and processes anymore?
        - Where do such standards and processes come from?
        - Do people perceive such standards and processes as beneficial or as burden due to overregulation?
        - Is software regularly checked throughout the process or is it just developed and passed on?
        - Which activities throughout the software development are perceived most beneficial and why?
    - How often does a software go through the agile improvement before being finalised?

  - Is a software ever finalised?
  - Does the model cover the agile development process?
  - What happens with software in use where changes in the earlier stages are necessary?
  - Who gives feedback on what and in which way?
  - To what extent can such feedback be used by developers?
  - What mechanism describes the work to do? (Backlog)
    - How is the backlog filled (which activities)?
    - What is part of the backlog?
- Work Pressure (if the concept is still unclear, conduct the ratio-exercise)
  - What aspects change work pressure?
    - To what extend can you increase work speed?
    - To what extend can you have overtime?
    - To what extend can you delay the work to be done? What happens when there is delay?
    - To what extend can there be constantly less work to do in order to decrease pressure?
    - Is staff hired in case of persistent too high work pressure?
    - Are there any other mechanisms that have an impact on work pressure?
  - How do people perceive work pressure?
  - Until what level would the people in the room perceive work pressure to be beneficial?
    - *Please draw this as a graph*
- Productivity (see also above for work pressure)
  - What aspects have an effect on productivity?
  - What is the normal productivity of a software developer?
- Parameters (if discussions on data or values come up, or if abundant time is available)
  - Avg. Amount of Software in earlier Stages (if these terms are accepted)
  - Avg. Amount of Software in later Stages (if these terms are accepted)
  - Time / Software or Time / Line of Code
  - Time / stage (requirement, design, testing, etc.)
  - Effect of tests on software quality
  - Time / Test
  - DevOps / Test

4.2 Errors:

Time available for Task:	14.15 - 14.30 (15 min)
Primary Nature of Task:	Divergent
Facilitator:	Jonas

- If the participants have not yet come to the topic of errors in software development (which is unlikely due to its connection to software quality), present the preliminary sub-model of Errors through the power point presentation
- Clarify for questions
- Put the paper-model on the large table between the participants and add it to the existing structure
  - Propose the connection from work speed to introduction of errors.
- Invite the participants to discuss, change, and adjust the model and delete parts of it

- Overall Model - Disconfirmation

  - What is missing in the picture?
  - What is wrong in the picture?
  - What would you like to change?
  - How are errors connected to the previous model on software development?
- Errors
  - May software move on in the development despite having known errors or is that software put on hold until the error is resolved?
    - Does this depend on the error?
  - Are errors from earlier stages automatically passed on to later stages?
  - May it happen that a software development project must be stopped and restarted due to unfixable errors?
  - What happens with errors in the end after releasing the software?
    - Vulnerability
  - Are errors and vulnerabilities fixed in such a way that future work profits from it or are only problems solved and future software (and architecture) is not adjusted?
    - Basically: Difference between fixing and error and just patching it.
- Tests
  - Which tests are conducted at what stage to find errors?
  - Do these tests take a considerable amount of time?
  - Do these tests have a considerable impact on software quality?
  - Does the difficulty of finding errors increase with the stages?
- Fixing
  - How is software fixed?
  - Do these tests activities take a considerable amount of time?
  - Do these tests have a considerable impact on software quality?
  - Does the difficulty of fixing errors increase with the stages?
- Parameters (if discussions on data or values come up, or if abundant time is available)
  - Average Amount of Known Errors in earlier stages (if these terms are accepted)
  - Average Amount of Known Errors in later stages (if these terms are accepted)
  - Based on the known amount and the test outcomes:
    - Guessed amount of Unknown Errors in earlier stages (if these terms are accepted)
    - Guessed amount of Unknown Errors in later stages (if these terms are accepted)
  - Errors / Software or Errors or Errors / Line of Code
  - Time / Fixing
  - Time / Testing
  - DevOps / Testing
  - DevOps / Fixing
  - Average Amount of accepted Errors

5. BREAK:

14.30 - 14.40 (10 min)

6. MODELLING - SOFTWARE DEVELOPMENT, ERRORS, AND DEVOPS II:

14.40 - 15.45 (65 min)

	Description
Roles	Facilitator: Jonas Assistant: Colleague
Aim	• Conduct group model building on the sub-models • Build the qualitative structure of the issues addresses by the sub-models at hand
Steps	(4.1 Software Development (most likely finished)) 6.1 Errors (and 4.2 if not finished) 6.2 DevOps (if little time available, and if possible both) 6.3 Bought / Third Party (if much time available, and if possible both)
Scripts	Causal mapping with seed structure, concept model, ratio exercise, Vennix, 1996
Notes	

6.1 Errors:

Time available for Task:	14.40 - 15.05 (25 min)
Primary Nature of Task:	Divergent
Facilitator:	Jonas

- Continue with the modelling exercise from before the break.
- For questions on errors, see under 4.2.

6.2 DevOps:

Time available for Task:	15.05 - 15.45 (40 min)
Primary Nature of Task:	Divergent
Facilitator:	Jonas

- If there is little time available and if the participants have not yet come to the topic of DevOps and Staff, present the preliminary sub-model of DevOps through the power point presentation
- Clarify for questions
- Put the paper-model on the large table between the participants. If there is enough space add it to the existing structure. Otherwise, hang the models on software development and errors clearly visible, so that the participants can refer to it.
  - If there are connections available in the previous models, propose to have those connections.
  - The most logical connection is from DevOps to Available DevOps for Development
- Invite the participants to discuss, change, and adjust the model and delete parts of it

- Overall Model - Disconfirmation
  - What is missing in the picture?
  - What is wrong in the picture?
  - What would you like to change?

    - How is DevOps connected to the previous model on software development?
- DevOps Skills
    - How do the overall skills of DevOps change?
        - Amount of staff
        - Experience (time in the organisation)
        - Knowledge (time as a developer)
        - Training
    - How are skills connected to productivity?
    - Is it common that developers have knowledge in security?
    - Do developers even think about security concerns or are they focussing on the functionality of the software?
- DevOps in Software Development
    - Next to skills, do other aspects, for instance motivation, need to be considered?
    - How is the staff connected to pressure?
    - What is the impact of DevOps on software quality?
- Parameters (if discussions on data or values come up, or if abundant time is available)
    - Time to familiarise with the organisation
    - Time of skills of natural decay
    - Effect of experience on skills
    - Effect of knowledge on skills
    - Amount of Developers
    - Hiring and firing rate
    - Average skill level

6.3 Third Party Software:

Time available for Task:	15.05 - 15.45 (40 min)
Primary Nature of Task:	Divergent
Facilitator:	Jonas

- If there is much time available, present the preliminary sub-model of Third Party Software through the power point presentation
- Clarify for questions
- Put the paper-model on the large table between the participants. If there is enough space add it to the existing structure. Otherwise, hang the models on software development and errors clearly visible, so that the participants can refer to it.
    - The model already suggests connections via schedule pressure and productivity. Clarify whether these connections are accurate.
- Invite the participants to discuss, change, and adjust the model and delete parts of it

- Overall Model - Disconfirmation
    - What is missing in the picture?
    - What is wrong in the picture?
    - What would you like to change?
    - Are the proposed connections accurate?
- Testing Third Party Software

- Why does testing of third party software takes place?
- How does testing of third party software takes place?
- Which tests are conducted?
- Do these tests take a considerable amount of time?
- Do these tests have a considerable impact on software quality?

- Patching, Software Quality and Exploitation
  - What happens if flaws are found in software?
  - Is software used despite of flaws?
  - What is the impact of flawed third party software on software quality and the possibility of exploitation?
  - Is software tested again after having been patched?
  - What happens in case of non-compatibility?
- Parameters (if discussions on data or values come up, or if abundant time is available)
  - Time / Test
  - DevOps / Test
  - Average Amount of accepted external Errors
  - Effect of tests on software quality
  - Effect of Flaws on software quality and possibilities of exploitation

7. CONCLUSION:

15.45 - 16.00 (15 min)

	Description
Roles	Facilitator: Jonas Assistant: Colleague
Aim	• Make sure that all participants agree with the outcomes of the session • Give an outlook for the next session
Steps	7.1 Review of the Session 7.2 Review of the Model 7.3 Next Steps
Scripts	Model review, next steps and closing
Notes	

7.1 Review of the Session:

Time available for Task:	15.45 - 15.50 (5 min)
Primary Nature of Task:	Presentation
Facilitator:	Jonas

- Aim of modelling workshop
  - *Improve software quality and thereby decrease possibility of exploitation by better knowing the interrelations within the organisation*
- Conceptual overview
  - Mention areas modelled
  - Connect to major challenges mentioned at the beginning

7.2 Review of the Model:

Time available for Task:	15.50 - 15.55 (5 min)
Primary Nature of Task:	Presentation
Facilitator:	Jonas

- Point out the feedback loops of each sub-model and emphasise the key insights
  - Software development & errors
  - DevOps
  - Third party software

7.3 Next Steps:

Time available for Task:	15.55 - 16.00 (5 min)
Primary Nature of Task:	Presentation
Facilitator:	Jonas

- Next session on Friday 24 March 2017
- Topic: Vulnerabilities and Adversarial Behaviour
- Session report will be send out on Monday 20.03.2017
- Questions?
- Thanking for the great participation

II. A. 2 Preparatory Script Group Model Building Session 2

OVERVIEW:

Location:	Conference Room within the Financial Organisation
Date:	24.03.2017
Topics:	Review First Session, Measuring Quality, Adversary Behaviour, Responsible Disclosure
Facilitator:	Jonas Matheus
Recorder:	Colleague
Participants:	Five with expertise in Ethical Hacking, Fraud, Penetration Testing, Responsible Disclosure, Software Development, and Vulnerability Scanning

Step	Facilitator	Assistant	Time	Min
Introduction	Jonas	Colleague	11.00 - 11.05	5
Revision • Major Challenges & Conceptual Overview • Model Revision by Reflector Feedback • Measuring Quality by Key Performance Indicators	Jonas	Colleague	11.05 - 11.35 11.05 - 11.10 11.10 - 11.25 11.25 - 11.35	30 5 15 10
Modelling • Vulnerabilities • Adversary Behaviour	Jonas	Colleague	11.35 - 12.30 11.35 - 11.50 11.50 - 12.30	55 15 40
Break			12.30 - 12.40	10
Modelling • Adversary Behaviour connected to Response and DevOps • Responsible Disclosure • Third Party Software (If time)	Jonas	Colleague	12.40 - 13.45 12.40 - 13.05 13.05 - 13.45	65 25 40
Conclusion	Jonas	Colleague	13.45 - 14.00	15

0. PREPARATION:

(420 min)
08.00 - 11.00 (180 min)

	Description
Roles	Facilitator: Jonas Assistant: Colleague
Aim	• Preparation of the session • Clarification among the facilitators • Preparation of the location
Steps	0.1 Script preparation 0.2 Preliminary model preparation in Vensim 0.3 Powerpoint presentation 0.4 Preliminary model preparation on paper (on site) 0.5 Clarification among facilitators 0.6 Preparation of location (on site)
Scripts	Scheduling the day, Logistics and room set up, Vennix, 1996

0.1 Script Preparation:

Time available for Task:	180 min
Primary Nature of Task:	Offline
Facilitator:	Jonas

- Look up all scripts on Scriptapedia and check for applicability
- Based on previous (successful) project, prepare new session-scripts

0.2 Preliminary Model in Vensim:

Time available for Task:	120 min

Primary Nature of Task:	Offline
Facilitator:	Jonas

- Focus on small models (Richardson, 2013; Vennix, 1996)
- Decide on “Concept Model” (quantified) or “Preliminary Model” (qualitative)
- Focus on software development, errors, DevOps staff, and third party software

0.3 Powerpoint Presentation:

Time available for Task:	60 min
Primary Nature of Task:	Offline
Facilitator:	Jonas

- Powerpoint presentation for guiding the participants through the workshop and presenting the conceptual model, system dynamics, and the preliminary models
- Based on previous (successful) project, prepare new presentation

0.4 Preliminary Model on Paper:

Time available for Task:	120 min (on site)
Primary Nature of Task:	Offline
Facilitator:	Jonas

- Draw the Vensim models on paper in the facilities of the organisation

0.5 Clarification and Planning among Facilitator and Recorder:

Time available for Task:	60 min
Primary Nature of Task:	Offline
Facilitator:	Jonas & Colleague

- Principal facilitator presents script to the assisting facilitator / recorder
- Discussion on unclear issues or aspects to be changed / improved

0.6 Preparation of Location:

Time available for Task:	60 min (on site)
Primary Nature of Task:	Offline
Facilitator:	Jonas & Colleague

Entirely based on the script on Logistics and Room Set Up

- “Arrange the table, chairs, and flip charts in the room in a manner conducive to upcoming activities and scripts. Let the participants sit in a semicircle facing either the wall where a model is projected, the white board, or the chalkboard.”
    - Around a big table, facing the screen, white board and windows
- “Arrange power cords, tables, and chairs for members not sitting at the table with participants (e.g., recorders, modellers, coaches). Secure any power cords and extension cables with tape to minimise the risk that people may trip.”

    - Power cords in the table, chairs available
- Arrange refreshments in a place that is convenient for participants to get up and access during the session.
    - Refreshments and snacks are provided by the organisation

1. INTRODUCTION:

11.00 - 11.05 (5 min)

	Description
Aim	Introduction of the Session & Aims of the day
Steps	Introduction
Scripts	
Notes	

Time available for Task:	11.00 - 11.05 (5 min)
Primary Nature of Task:	Presentation

- Welcoming participants and thanking for the active collaboration in the previous session.
- Aim of the session:
    - Review and some discussion of the outcomes of the previous session
    - This session, focus on external influence on the organisation
- Agenda Presentation & Parking Lot Explanation
- Questions

2. REVISION:

11.05 - 11.35 (30 min)

	Description
Aim	• Revision and shared agreement on previous session • Elicitation of KPIs
Steps	2.1 Major Challenges & Conceptual Overview 2.2 Model Revision by Reflector Feedback 2.3 Measuring Quality by Key Performance Indicators
Scripts	Reflector Feedback, NGT, Variable Elicitation
Notes	

2.1 Major Challenges & Conceptual Overview:

Time available for Task:	11.05 - 11.10 (5 min)
Primary Nature of Task:	Presentation & Convergent

- Presentation of the adjusted diagram of the submodels.
- Make sure that this is just a rough connection and of course very much simplified.
- Reveal step by step the included major challenges and briefly describe them, emphasise…
    - Internal Software Development -> Revision (2.2)
        - Monetary Focus creates lack of security (ASAP)
        - Possible connection to Staff & Training
    - Vulnerability and Incidents
        - Adversary advantage and delays
        - Working on it this session
    - Overarching picture
        - Not directly covered, but rather implicitly in the model (e.g. dependencies, double work)
        - Hope that this project helps to get a more holistic view
    - "How to measure quality?" -> Revision (2.3)
- If necessary, questions and comments for clarification, not for discussion, are possible.
- Questions

2.2 Model Revision by Reflector Feedback:

Time available for Task: 11.10 - 11.25 (15 min)
Primary Nature of Task: Presentation & Convergent

- Presentation of the three versions of the model:
    - Copied model: very complex and detailed
    - Intermediate model: more clear, yet still way too difficult
    - Final model: still quite some complexity in order to actually cover the richness of the picture
- Reveal loop by loop the model and explain the dynamics.
    - Lack of discipline: Why? Security fatigue?
    - External Pressure: Why? Monetary reasons?
- If necessary, questions and comments for clarification, not for discussion, are possible.
- Questions

2.3 Measuring Quality by Key Performance Indicators:

Time available for Task: 11.25 - 11.35 (15 min)
Primary Nature of Task: Divergent & Convergent

- Present a slide only stating: *"How to measure quality?"*
- Refer to the major challenges mentioned at the beginning
- Ask the participant to write down in two to three words which indicators they would use for measuring whether software quality regarding internal software development has improved or deteriorated
- Refer also to the model and emphasise that we have a non-technical focus
- Facilitator asks the participants to present one of their ideas within 20 to 30 seconds in a round-robin fashion.
- If necessary, questions and comments for clarification, not for discussion, are possible.
- Questions

3. MODELLING:

11.35 - 12.30 (55 min)

	Description
Aim	• Present the two submodels • Conduct group model building on these two submodels • Build the qualitative structure of the issues addresses by the submodels at hand
Steps	3.1 Software Vulnerabilities 3.2 Adversary Behaviour
Scripts	Causal mapping with seed structure, concept model, ratio exercise, initiating and elaborating a causal loop diagram, Vennix, 1996
Notes	

3.1 Software Vulnerabilities:

Time available for Task:	11.35 - 11.50 (15 min)
Primary Nature of Task:	Divergent

- Present the preliminary submodel of software vulnerabilities through the power point presentation (Software vulnerability = weak spot in a software after release)
- Clarify for questions
- Put the paper-model on the large table between the participants
- Invite the participants to discuss, change, and adjust the model and delete parts of it
    - Invite the participants to first discuss causes for software vulnerabilities and the handling (10 min)
    - Invite the participants to then discuss consequences of software vulnerabilities (5 min)
- In case a discussion does not really come up, the following questions may help to get the participants talk about the topic and interact with the model:
- Overall Model - Disconfirmation
    - What is missing in the picture?
    - What is wrong in the picture?
    - What would you like to change?
- Causes for Vulnerabilities and Handling
    - Last time we discussed errors in the internally developed software and connected those to incidents. How does this match with the picture of vulnerabilities?
    - Are there any further elements that cause vulnerabilities?
    - What happens with vulnerabilities?
    - How are vulnerabilities detected and resolved?
    - Who detects and resolves vulnerabilities? (DevOps?)
    - Is it planned in the Sprint Backlog to handle (detect and resolve) vulnerabilities?
        - Does it a critical vulnerability cause a major disruption in the backlog?
    - How long does it take to solve vulnerabilities?
- Consequences of Vulnerabilities
    - What happens if software vulnerabilities are in place?
        - Reporting from RD
        - Exploitation from Adversary
- Once it comes to adversary: add new structure of how an adversary behaves

3.2 Adversary Behaviour:

Time available for Task:	11.50 - 12.30 (40 min)
Primary Nature of Task:	Divergent

- Once it comes to adversary: Stop the discussion and briefly present the submodel on adversary behaviour via the presentation
- Clarify for questions
- Put the paper-model on the large table between the participants and add it to the existing structure
    - Propose the connection from vulnerabilities to risk
- Invite the participants to discuss, change, and adjust the model and delete parts of it

- Overall Model - Disconfirmation
    - What is missing in the picture?
    - What is wrong in the picture?
    - What would you like to change?
    - Is the connection from vulnerabilities to risk appropriate?
- Risk, Threat and Vulnerabilities
    - How do you define risk in the organisation?
        - Is it suitable to say that vulnerability & threat define risk?
    - What constitutes a threat?
        - Is is suitable to say that attack preparation chain (based on motivation and resources + skills) defines a threat?
    - How do you deal with risk?
    - Do you measure risk?
    - Do you recognise if risk changes?
    - What do you do if risk changes?
- Adversary Dynamics
    - Does a successful attacker comes again?
    - What does an unsuccessful attacker do?
    - Do attackers aim for the low hanging fruits?
    - Do attacker get to know about the success of others?
    - What else leads to a decreasing number of attacks?
- Adversary Productivity
    - What constitutes the productivity of an adversary?
        - Resources (people, money, tools)
        - Experience / Maturity / Skills
        - Motivation / Threat
    - How does an adversary's productivity changes?
- Hacking & Advanced Persistent Threat (APT)
    - Is this representation of an APT suitable?
    - Does a hacker actually follow these steps?
    - Is a step missing in the picture?
    - What happens when the adversary stays longer within the system?
    - Is it easier or more difficult to detect an adversary when s/he stays longer in the system?
    - Since people always talk about staying under the radar, how much can an adversary learn per round in the system without being recognised?

- Detection of Activities & Recognition of Incident
    - How do you recognise an adversary hacking the system?
    - How do you recognise an adversary in case of an APT hacking the system?
    - Which possibilities has an adversary to cover his/her actions?
    - Is the detection conducted with automated tools or does it cost staff?
        - In case of staff: Who is working on detecting activities?
    - How do you recognise an incident?
        - Incident here understood as doing something and not only staying in the system
    - Is the detection of an incident conducted with automated tools or does it cost staff?
        - In case of staff: Who is working on detecting activities?
- Mitigate the Attack
    - What do you do in case of detecting activities?
    - What do you do in case of an incident?
    - See next section: Adversary behaviour connected to Response and DevOps

4. BREAK:

12.30 - 12.40 (10 min)

5. MODELLING:

12.40 - 13.45 (65 min)

	Description
Aim	• Continue working on the previous models • Present at least one further submodel • Conduct group model building on these submodels • Build the qualitative structure of the issues addresses by the submodels at hand
Steps	5.1 Response 5.2 Responsible Disclosure 5.3 Bought / Third Party (if much time available)
Scripts	Causal mapping with seed structure, concept model, ratio exercise, Vennix, 1996
Notes	

5.1 Response:

Time available for Task:	12.40 - 13.10 (30 min)
Primary Nature of Task:	Divergent

- Continue with the modelling exercise from before the break - focus on Response.
- Mitigation
    - What do you do in case of detecting activities?
    - What do you do in case of an incident?
    - How long can a mitigation last?
    - How does a mitigation actually takes place? Do you “kick out” the hacker? Do you change the code? Do you shut down the system? Do you close parts of the system?
    - What has an influence on the effectiveness of a response?
    - How do you know that a hacker is not in the system anymore?

- Staff
    - Who mitigates an attack?
    - What happens with the work these people have to do?
    - What happens with the backlog of the teams mitigating an attack?
    - How long can you delay normal work for mitigating an attack?
    - Could you say: The more staff, the more effective in responding?
- Incident
    - Do you have to report an incident?
    - What are the consequences of an incident?
    - What are the consequences of a successful attack?
    - Do you have to pay customers for successful attacks? (Information and / or monetary loss)
    - What do you do after an incident?
    - Last time we said that we create new rules after an incident. Does this apply here? What kind of rules may be created? Incident, development, testing, HR, deterrence and punishment?

5.2 Responsible Disclosure:

Time available for Task:	13.10 - 13.45 (35 min)
Primary Nature of Task:	Divergent

- Present the preliminary submodel on responsible disclosure through the power point presentation
- Explain that we look at responsible disclosure for adding a second view on external interaction (here ethical hackers from outside)
- Clarify for questions
- Put the paper-model on the large table between the participants
- Invite the participants to discuss, change, and adjust the model and delete parts of it

- In case a discussion does not really come up, the following questions may help to get the participants talk about the topic and interact with the model:

- Overall Model - Disconfirmation
    - What is missing in the picture?
    - What is wrong in the picture?
    - What would you like to change?
- Responsible Disclosure
    - What is the aim of having RD?
    - How many ethical hackers are actually attracted by RD programmes?
    - What has an impact on ethical hackers taking part in the organisation's RD?
        - Vulnerabilities
        - Payment / Income
        - Status / Reputation of the organisation
        - Communication and being on time
    - If the RD (particularly status / reputation and vulnerabilities) of an organisation attracts ethical hackers, does it also attract malicious hackers?
    - To what extend is communication important for RD?

  - What happens if known vulnerabilities are found?
  - What defines the effectiveness of RD programmes?
  - What is the average amount of vulnerabilities found by ethical hackers?
  - Critical: What happens if ethical hackers only search for the low hanging fruits?
    - Aren’t those anyways only those vulnerabilities that could have been fixed at the outset?
    - May that create a wrong incentive to just not pay attention to errors since they are going to be found anyways?

5.3 Third Party Software:

Time available for Task:	If time available
Primary Nature of Task:	Divergent

- If there is much time available, present the preliminary submodel of Third Party Software through the power point presentation
- Clarify for questions
- Put the paper-model on the large table between the participants. If there is enough space add it to the existing structure. Otherwise, hang the models on software development and errors clearly visible, so that the participants can refer to it.
  - The model already suggests connections via schedule pressure and productivity. Clarify whether these connections are accurate.
- Invite the participants to discuss, change, and adjust the model and delete parts of it

- Overall Model - Disconfirmation
  - What is missing in the picture?
  - What is wrong in the picture?
  - What would you like to change?
  - Are the proposed connections accurate?
- Testing Third Party Software
  - Is actually all external software tested?
  - Who is actually conducting these tests?
  - Why does testing of third party software takes place?
  - How does testing of third party software takes place?
  - Which tests are conducted?
  - Do these tests take a considerable amount of time?
  - Do these tests have a considerable impact on software quality?
- Patching, Software Quality and Exploitation
  - What happens if flaws are found in software?
  - Is software used despite of flaws?
  - Is software used without being tested?
  - What is the impact of flawed third party software on software quality and the possibility of exploitation?
  - Is software tested again after having been patched?
  - What happens in case of non-compatibility?

6. CONCLUSION

13.45 - 14.00 (15 min)

	Description
Aim	• Review the session regarding aims and connection to previous session • Review the models built in the session • Give a claret outlook for the upcoming session
Steps	6.1 Review of the Session 6.2 Review of the Model 6.3 Next Steps
Scripts	Model review, next steps and closing
Notes	

6.1 Review of the Session:

Time available for Task:	13.45 - 13.50 (5 min)
Primary Nature of Task:	Presentation

- Aim of modelling session
  - Review and discussion of the outcomes of the previous session
  - Focus on external influence on the organisation
- Conceptual overview
  - Mention areas modelled
  - Connect to major challenges mentioned at the beginning
    - Money-driven better before worse approach with focus on functionality
    - Adversary advantage
    - Lack of overarching picture and understanding
    - How to measure quality?

6.2 Review of the Model:

Time available for Task:	13.50 - 13.55 (5 min)
Primary Nature of Task:	Presentation

- Point out the mechanisms of each submodel and emphasise the key insights:
  - Vulnerabilities
  - Adversary Behaviour & Response
  - Responsible Disclosure
  - Third Party Software

6.3 Next Steps:

Time available for Task:	13.55 - 14.00 (5 min)
Primary Nature of Task:	Presentation

- Next session on Tuesday 28 March 2017
- Topics:
  - Revision session 2 including KPIs
  - DevOps Training & Awareness, Third Party Software
  - Walk Through entire model
  - Policy discussion
- Session report will be send out on Monday 27.03.2017
- Questions?
- Thanking for the great participation

II. A. 3 Preparatory Script Group Model Building Session 3

OVERVIEW:

Location: Conference Room within the Financial Organisation

Date: 28.03.2017

Topics: Review Second Session, Measuring Quality, DevOps Training, Third Party Software, Model Walkthrough, Policy Discussion

Facilitator: Jonas Matheus
Recorder: Colleague

Participants: Five with expertise in Ethical Hacking, Fraud, Penetration Testing, Responsible Disclosure, Software Development, and Vulnerability Scanning

Step	Facilitator	Assistant	Time	Min
Introduction	Jonas	Colleague	12.00 - 12.05	5
Revision • Model Revision by Reflect Feedback • Measuring Quality by Key Performance Indicators	Jonas	Colleague	12.05 - 12.30 12.05 - 12.20 12.20 - 12.30	25 15 10
Modelling • Third Party Software • DevOps Training and Awareness	Jonas	Colleague	12.30 - 13.20 12.30 - 12.50 12.50 - 13.20	50 20 30
Break			13.20 - 13.30	10
Model Walkthrough • Internal Software Development & Third Party Software • DevOps & Training • Vulnerabilities • Adversary Behaviour & Response • Responsible Disclosure	Jonas	Colleague	13.30 - 14.20 13.30 - 13.45 13.45 - 13.55 13.55 - 14.00 14.00 - 14.15 14.15 - 14.20	50 15 10 5 15 5
Improving Software Quality: Policies • Collect Ideas via NGT • Present and Discuss Ideas • Vote about Ideas	Jonas	Colleague	14.20 - 14.50 14.20 - 14.30 14.30 - 14.45 14.45 - 14.50	30 10 15 5
Conclusion	Jonas	Colleague	14.50 - 15.00	10

0. PREPARATION:

(420 min)
09.00 - 12.00 (180 min)

	Description
Roles	Facilitator: Jonas Assistant: Colleague
Aim	• Preparation of the session • Clarification among the facilitators • Preparation of the location
Steps	0.1 Script preparation 0.2 Preliminary model preparation in Vensim 0.3 Powerpoint presentation 0.4 Preliminary model preparation on paper (on site) 0.5 Clarification among facilitators 0.6 Preparation of location (on site)
Scripts	Scheduling the day, Logistics and room set up, Vennix, 1996

0.1 Script Preparation:

Time available for Task:	180 min
Primary Nature of Task:	Offline
Facilitator:	Jonas

- Look up all scripts on Scriptapedia and check for applicability
- Based on previous (successful) project, prepare new session-scripts

0.2 Preliminary Model in Vensim:

Time available for Task:	120 min
Primary Nature of Task:	Offline
Facilitator:	Jonas

- Focus on small models (Richardson, 2013; Vennix, 1996)
- Decide on “Concept Model” (quantified) or “Preliminary Model” (qualitative)
- Focus on software development, errors, DevOps staff, and third party software

0.3 Powerpoint Presentation:

Time available for Task:	60 min
Primary Nature of Task:	Offline
Facilitator:	Jonas

- Powerpoint presentation for guiding the participants through the workshop and presenting the conceptual model, system dynamics, and the preliminary models
- Based on previous (successful) project, prepare new presentation

0.4 Preliminary Model on Paper:

Time available for Task:	120 min (on site)
Primary Nature of Task:	Offline
Facilitator:	Jonas

- Draw the Vensim models on paper in the facilities of the organisation

0.5 Clarification and Planning among Facilitator and Recorder:

Time available for Task:	60 min
Primary Nature of Task:	Offline
Facilitator:	Jonas & Colleague

- Principal facilitator presents script to the assisting facilitator / recorder
- Discussion on unclear issues or aspects to be changed / improved

0.6 Preparation of Location:

Time available for Task:	60 min (on site)
Primary Nature of Task:	Offline
Facilitator:	Jonas & Colleague

Entirely based on the script on Logistics and Room Set Up

- “Arrange the table, chairs, and flip charts in the room in a manner conducive to upcoming activities and scripts. Let the participants sit in a semicircle facing either the wall where a model is projected, the white board, or the chalkboard.”
    - Around a big table, facing the screen, white board and windows
- “Arrange power cords, tables, and chairs for members not sitting at the table with participants (e.g., recorders, modellers, coaches). Secure any power cords and extension cables with tape to minimise the risk that people may trip.”
    - Power cords in the table, chairs available
- Arrange refreshments in a place that is convenient for participants to get up and access during the session.
    - Refreshments and snacks are provided by the organisation

1. INTRODUCTION

12.00 - 12.05 (5 min)

	Description
Aim	Introduction of the Session & Aims of the day
Steps	Introduction
Scripts	
Notes	

Time available for Task:	12.00 - 12.05 (5 min)
Primary Nature of Task:	Presentation

- Welcoming participants and thanking for the active collaboration in the previous session.
- Aim of the session:
    - Review and some discussion of the outcomes of the previous session
    - Modelling focus on Third Party Software & Training and Awareness
    - *Overcome specialised islands and get the big picture*
    - *Improving software quality*
- Agenda Presentation & Parking Lot Explanation
- Questions

2. REVISION

12.05 - 12.30 (25 min)

	Description
Aim	• Revision and shared agreement on previous session • Elicitation of KPIs
Steps	2.1 Model Revision 2.2 Key Performance Indicators
Scripts	Reflector Feedback, NGT, Variable Elicitation
Notes	

2.1 Model Revision by Reflector Feedback:

Time available for Task:	12.05 - 12.20 (15 min)
Primary Nature of Task:	Presentation & Convergent

- Presentation of the three versions of the model:
    - Copied model: very complex and detailed
    - Intermediate model: more clear, yet still way too difficult
    - Final model: still quite some complexity in order to actually cover the richness of the picture
- Reveal loop by loop the model and explain the dynamics.
- If necessary, questions and comments for clarification, not for discussion, are possible.
- Questions

2.2 Measuring Quality by Key Performance Indicators:

Time available for Task:	11.25 - 11.35 (15 min)
Primary Nature of Task:	Divergent & Convergent

- Present a slide only stating: *"How to measure quality?"*
- Refer to the previous session and to the models built so far
- Ask the participant to write down in two to three words which indicators they would use for measuring whether software quality regarding vulnerabilities, adversary behaviour, and responsible disclosure has improved or deteriorated

- Refer also to the model and emphasise that we have a non-technical focus
- Facilitator asks the participants to present one of their ideas within 20 to 30 seconds in a round-robin fashion.
- If necessary, questions and comments for clarification, not for discussion, are possible.
- Questions

3. MODELLING

12.30 - 13.20 (50 min)

	Description
Aim	• Present the two submodels • Conduct group model building on these two submodels • Build the qualitative structure of the issues addresses by the submodels at hand
Steps	3.1 Third Party Software 3.2 Software Developer Training and Awareness
Scripts	Causal mapping with seed structure, concept model, ratio exercise, initiating and elaborating a causal loop diagram, Vennix, 1996
Notes	

3.1 Third Party Software:

Time available for Task:	12.30 - 12.50 (20 min)
Primary Nature of Task:	Divergent

- Present the preliminary submodel of Third Party Software through the power point presentation
- Clarify for questions
- Put the paper-model on the large table between the participants.
- Invite the participants to discuss, change, and adjust the model and delete parts of it
- In case a discussion does not really come up, the following questions may help to get the participants talk about the topic and interact with the model:

- Overall Model - Disconfirmation
  - What is missing in the picture?
  - What is wrong in the picture?
  - What would you like to change?
  - Are the proposed connections accurate?
- Testing Third Party Software
  - Is actually all external software tested?
  - Who is actually conducting these tests?
  - Why does testing of third party software takes place?
  - How does testing of third party software takes place?
  - Which tests are conducted?
  - Do these tests take a considerable amount of time?

  - Do these tests have a considerable impact on software quality?
- Patching, Software Quality and Exploitation
  - What happens if flaws are found in software?
  - Is software used despite of flaws?
  - Is software used without being tested?
  - What is the impact of flawed third party software on software quality and the possibility of exploitation?
  - Is software tested again after having been patched?
  - What happens in case of non-compatibility?

3.2 DevOps Training and Awareness:

Time available for Task:	12.50 - 13.20 (30 min)
Primary Nature of Task:	Divergent

- Present the preliminary submodel of DevOps Training and Awareness through the power point presentation
- Clarify for questions
- Put the paper-model on the large table between the participants.
- Invite the participants to discuss, change, and adjust the model and delete parts of it
- In case a discussion does not really come up, the following questions may help to get the participants talk about the topic and interact with the model:

- Overall Model - Disconfirmation
  - What is missing in the picture?
  - What is wrong in the picture?
  - What would you like to change?
  - Are the proposed connections accurate?
  - How do maturity, security awareness, awareness about security trainings, and security trainings affect each other?
- Maturity
  - What influences DevOps Maturity?
    - Amount of staff?
    - Experience (time in the organisation)?
    - Knowledge (time as a DevOp)?
    - Training?
    - Awareness?
  - Does maturity indicate knowledge in security?
  - Is it common that DevOps have knowledge in security?
  - Do developers even think about security concerns or are they focussing on the functionality of the software?
- Security Training
  - Are DevOps trained in security
  - How are DevOps trained?
  - How often are DevOps trained?
  - How long does a training take?
  - What are the benefits of training?

    - What are downsides of training?
    - Is the effect of training assessed?
- Awareness about Security Training
    - Do DevOps know about training?
    - How do DevOps get to know about training?
    - Which mechanisms are there next to just talking with colleagues?
    - Is awareness about security training even important? Or is it logical and everybody knows about such trainings?
- Awareness about Security
    - Is there a difference between the awareness about security training and the awareness about security itself?
    - What does security awareness mean?
    - Is security awareness a problem?
    - How does one change security awareness?
    - Are there other ways to influence security awareness than by training?
    - Does security fatigue or overtraining or oversecuritisation exist?

4. BREAK:

13.20 - 13.30 (10 min)

5. MODEL WALKTHROUGH:

13.30 - 14.20 (50 min)

	Description
Aim	• Revision of and shared agreement on the previous modelling results of all sessions • Revision of and shared agreement on the links between the different submodels • Revision of and shared agreement on the critical areas of the overall mode
Steps	• Software: Internal Software Development & Third Party Software 15 min • Staff: DevOps Training and Awareness 10 min • Vulnerabilities 5 min • External Interaction 1: Adversary Behaviour and Response 15 min • External Interaction 2: Responsible Disclosure 5 min
Scripts	Model Review; Reflector Feedback, Disconfirmatory Interview
Notes	

Time available for Task: 13.30 - 14.10 (40 min)
Primary Nature of Task: Presentation & Convergent

- Presentation of the overall model:
    - Make the aims clear (see above)
    - Make clear that the results of this session are not included
    - Ask the participants to strongly question the results
- Reveal loop by loop the model and explain the dynamics.
- Discuss about all aspects the participants bring up. Include the participants by continuously probing for disconfirmation (use techniques described in Andersen et al. 2013)
- Questions

6. IMPROVING SOFTWARE QUALITY: POLICIES

14.20 - 14.50 (30 min)

	Description
Aim	• Find suitable Policies • Get an idea about importance of Policies
Steps	6.1 Policy Elicitation 6.2 Presentation, Clarification, and Discussion of Policies 6.3 Vote about Policies in the Model and on the Policies
Scripts	NGT, Voting with Dots
Notes	

6.1 Policy Elicitation:

Time available for Task:	14.20 - 14.30 (10 min)
Primary Nature of Task:	Divergent

- Introduction and aim (see above) of the exercise
- Present a slide only stating: *“How to improve quality?”*
- Ask the participant to write down in two to three words which policies they would use for improving software quality.
- Refer to all the previous models, the discussions and to the measurement criteria which were collected in the second and third (this) session.
    - Have these criteria ready on the wall and uncover them.
    - Have the overall Model on the screen.
- If necessary, questions and comments for clarification, not for discussion, are possible.
- Questions

6.2 Presentation, Clarification, and Discussion of Policies:

Time available for Task:	14.30 - 14.45 (15 min)
Primary Nature of Task:	Convergent

- Facilitator asks the participants to present one of their ideas within 20 to 30 seconds in a round-robin fashion. The facilitator clusters the policies on the wall.
- If the policy is not clear to all, the respective participant further clarifies the policy. Make sure each policy is well understood. No discussion of the policy at that point. Refer to the end of the “round” when each participant has proposed one idea.
- After one round, all ideas are discussed. The facilitator takes care that all participants are actively involved in the discussion. If that is not the case, the facilitator directly involves the non-active participants via questions, etc.
- There are either as many “rounds” as there are ideas, or until the time is over. A round should not be more than 5 minutes.
- Questions

6.3 Vote about Policies in the Model and on Policies:

Time available for Task:	14.45 - 14.50 (5 min)
Primary Nature of Task:	Ranking & Convergent

- Each of the participants receives a coloured pen.
- The facilitator explains that the participants have to vote about the best policy options.
- The proposed policies are clustered on the wall.
- Additionally, a printed version (A0) of the model used for the model walkthrough is set on the wall.
- Each of the participants has five points to make with the received pen on…
    - … the policies on the wall,
    - … the model which was used for the model walk through
- For the policy the participants opt for a specific policy.
- For the model, the participants indicate which aspect / area they perceive as most important.
- While the participants have to make a dot on the policies, they are allowed to circle an area in the model (emphasise that this should be a "conservative" circle, not the entire model or submodel).
- Questions

7. CONCLUSION

14.50 - 15.00 (10 min)

	Description
Aim	• Review the session regarding aims and connection to previous session • Review the models built in the session • Give a claret outlook for the upcoming session
Steps	6.1 Review of the Session 6.2 Review of the Workshops 6.3 Next Steps
Scripts	Model review, next steps and closing
Notes	

7.1 Review of the Session:

Time available for Task:	14.50 - 14.52 (2 min)
Primary Nature of Task:	Presentation
Facilitator:	Jonas

- Aim of modelling workshop
    - *Finalise last submodels*
    - *Review overall model*
    - *Vote on Policies*

7.2 Review of all Workshops:

Time available for Task:	14.52 - 14.57 (5 min)
Primary Nature of Task:	Presentation
Facilitator:	Jonas

- Major Challenges
- Conceptual Model
- Models
- Overall Model
- Measuring Quality and Policies for Quality Improvement

7.3 Next Steps:

Time available for Task:	14.57 - 15.00 (3 min)
Primary Nature of Task:	Presentation
Facilitator:	Jonas

- Validations will follow
- Session report will be send out on 05.04.2017.
- Workshop report with overall model and explanations will be send out after the validation session on Monday 10.04.2017
- Questions?
- Thanking for the great participation

II. B Qualitative Data Analysis

This study draws on the insights from a wide range of data sources. While the causal diagrams from the group model building workshops represent the major data source of this study (4, I), further qualitative data, such as notes taken during the modelling sessions (II. C), observations, conversations, and unstructured interviews (II. D), was used to round off the findings. Particularly in the results-section above, the triangulated data was used to underline the findings from the group model building workshops and to give further meaning and context to the insights, thereby, enriching the construct and internal validity of the research (Thurmond, 2001; Yin, 2014).

As common in qualitative research and case studies, the qualitative data was subject to coding and subsequent analysis through categorising reoccurring patterns and building explanations from it (Merriam, 2009; Yin, 2014). According to Merriam, "coding is nothing more than assigning some sort of shorthand designation to various aspects of your data so that you can easily retrieve specific pieces of the data. The designations can be single words, letters, numbers, phrases, colors, or combinations of these" (Merriam, 2009, p. 173). Combining the different codes and searching for patterns across the different data leads to the creation of categories or themes. In other words, different individual information that fall under the same umbrella are clustered into groups that represent abstractions derived from the data (Merriam, 2009). Merriam describes this exercise of data analysis as "making sense out of the data" which essentially means to address the research question (2009, p. 175f.). While data analysis is generally entirely inductive at the beginning, it becomes more and more deductive over time as the research proceeds (Figure II.B.1). In this context the following two aspects are noteworthy for the case study at hand: First, case study research is always guided by preliminary theory or theoretical proposition related to the topic of investigation, for instance, derived from scientific literature. While other qualitative approaches, such as ethnographic studies or grounded theory, may only be informed by literature or even deliberately avoid having such guidelines to obviate biases when collecting and analysing data, particularly

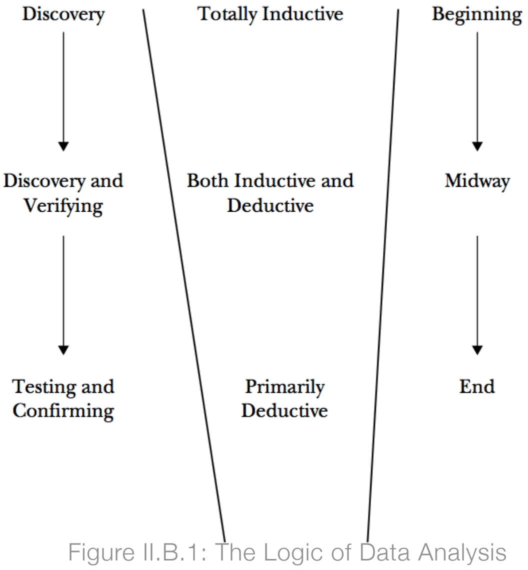


Figure II.B.1: The Logic of Data Analysis (Merriam, 2009, p. 184).

single case studies rely on the guidance from and connection to literature when collecting and analysing data (Merriam, 2009; Yin, 2014). As explained in the methodology-section, the case study at hand was induced by the phenomena of growing numbers of software vulnerabilities and successful cyber attacks, and guided by the scientific literature from several fields. As a consequence, the process of coding and categorising was automatically not totally inductive as described by Merriam (2009), but rather both inductive and deductive as theoretical elements from the reviewed literature supported and guided the data collection and analysis. Additionally, since data is collected for setting the scene of data analysis, and data analysis basically means addressing the research question, by their very nature the activities of coding and categorising are influenced by the research question. Second, the supporting qualitative data was collected and analysed during and mainly after the group model building workshops which represent the main data source of the case study. Hence, when the researcher started to analyse the larger chunk of data (only five conversations, unstructured interviews, and observations were made prior to the workshops), the study was already midway in its process which allowed the researcher to work in both ways, inductively and deductively, with the qualitative data (Merriam, 2009), as indicated in Figure II.B.1 above. The categories were developed following best practice described in research methodology. The number of categories, for instance, was kept large enough to address the research question and incorporate important findings from the study, but small enough to still handle the categories (Merriam, 2009).

II. Table 1 shows the codes and categories which were applied in this study. The table also indicates whether the code or category was derived inductively by coding and comparing the data, deductively by referring to the theoretical insights from the reviewed literature, by the research question, or by a combination of the different approaches. As common in qualitative research and particularly in case studies, the insights derived from analysing the data were integrated and abstracted in an effort of going beyond the data itself. According to Merriam, “one of the best ways to try this out is to visualize how the categories work together. A model is just that—a visual presentation of how abstract concepts (categories) are related to one another” (Merriam, 2009, p. 189). Along the same line, the insights derived from analysing the data were used to refine the causal diagrams developed in the group model building workshops. The final version of the causal diagram is shown in the results-section above. As common in case study research, in the discussion-section the insights were further generalised in an effort to make some first steps of building theory on vulnerability dynamics which goes beyond the case of software development and cyber security (Merriam, 2009; Yin, 2014).

II. Table 1: Summary of Codes and Categories applied in the Analysis of the Qualitative Case Data

Categories and Codes	Introduction
Software Development	Research Question; Inductive; Deductive
Software / Software Development / Software Operations / etc.	Research Question; Inductive; Deductive
Agile / Agile Approaches / Agility / Flexibility / Adaptation / etc.	Research Question; Inductive; Deductive
Productivity / Velocity / Disruption	Inductive
Backlog / Sprint Backlog / etc.	Inductive
Secure Software Development / Software Security	Research Question; Inductive; Deductive
Pressure	Research Question; Deductive; later confirmed and strengthened inductively
Stress / Time Pressure / Market Pressure / Workload / Work Ratio, etc.	Deductive
Curt, Brusque, Grumpy, Sharp Behaviour / etc.	Inductive
Observational Signs, e.g., eating at work / looking tired / etc.	Inductive
Trade-Off / Tension / Decision about Functionality and Security	Inductive
Self-Organisation / Push and Pull / (Strategic) Delay / etc.	Inductive
Emphasis of Business Orientation / Efficiency / etc.	Inductive
Risk / Business Risk / Security Risk / Financial Risk / Cost	Inductive
Short-Term / Short-Run / Long-Term / Long-Run	Inductive
Defects and Vulnerabilities	Research Question; Inductive; later confirmed and strengthened deductively
Error / Mistake / Defect / Flaw / Bug	Inductive; Deductive
Vulnerability / Vulnerable / Software Vulnerability	Research Question; Inductive; Deductive
Criticality / Severity (i.e., low, medium, high, critical)	Inductive
Numbers of Vulnerabilities	Research Question; Inductive
Unknown and Known	Inductive; later Deductive
Test / Fix / Resolve / Mean Time to Resolve / etc.	Inductive; Deductive
Maturity and Training	Inductive; later confirmed and strengthened deductively
DevOps / Engineers / Developers / etc.	Inductive; Deductive
Maturity / Skills / Proficiency / etc.	Inductive
Awareness / Security Awareness / Awareness Culture	Inductive
Training / Awareness Training / Skills Training / Process Improvement	Inductive; later Deductive
Layoffs / Restructuring / etc.	Inductive
Security Measures	Inductive; Deductive
Regulations / Policies / Guidelines / Standards	Inductive
Overregulation	Deductive; Inductive

II. Table 1: Summary of Codes and Categories applied in the Analysis of the Qualitative Case Data

Categories and Codes	Introduction
Security Approaches (e.g., Pentest, Responsible Disclosure, Red Team)	Inductive
Increase Team Size / Change Teams / etc.	Inductive
Response	Inductive
Problem Awareness / Systems Thinking	Inductive
Training & Awareness are categorised under Maturity & Training	*Inductive*
Development Support and Tools are categorised under Technology	*Inductive*
Adversarial Behaviour and Cyber Attacks	Research Question; Inductive
Hacking / Steps in Hacking / Hacking Process	Inductive
Exploitation (*Software Vulnerabilities and Vulnerability are categorised under Defects and Vulnerabilities)*	Research Question; Inductive; Deductive
Cyber Attacks / Cyber Criminal / Cyber Adversary	Research Question; Inductive; Deductive
Business Case / Adversary Strategy / Adversary Organisation / etc.	Inductive
Escalation / Escalatory Pattern / Arms Race / etc.	Inductive; Deductive
Adversary Learning / Capability / Effectiveness / Resources / etc.	Inductive
Malware / Malware Attacks	Inductive; later Deductive
Aspects of Perception of Management and Employees	Inductive; later confirmed and strengthened deductively
Managerial Opinion on Employees (e.g., lack of compliance)	Inductive; later Deductive
Managerial Indication of Problems / Understanding of Situation	Inductive; later Deductive
Decision Making Activities with emphasis of Business Orientation / Efficiency / etc. are categorised under Pressure	*Inductive*
Employee on Management and Business	Inductive
Employee Indication of Problems / Understanding of Situation	Inductive; later Deductive
Expression of Stress / etc. are categorised under Pressure	*Inductive*
Technology and Innovation	Inductive
Technology / Technical Solution / Technical Problem / etc.	Inductive
Innovation / Change in Future / etc.	Inductive
Rival Theories (not a category for results in itself)	*Inductive; Deductive*
Anything related to the category of Technology and Innovation	*Inductive; Deductive*
Dependencies / Politics / etc.	*Indcutive*
Third Party Software	*Inductive*
Anything related to the code of Overregulation in Security Measures	*Deductive; Inductive*
Future Research (not a category for results in itself)	*Inductive*
Economics / Financials / Costs / Benefits / etc.	*Inductive; later Deductive*
Quantification / Simulation / etc.	*Inductive; Deductive*
Strength of Mechanism / Probability of Occurrence / etc.	*Inductive*
Standardised Solutions & Technology	*Inductive*
Improvement	*Inductive*
Validity / Generalisation / Test Theory / etc.	*Deductive*

II. C Documentation Group Model Building

While it is common in qualitative research and participatory approaches of knowledge elicitation to rely on video and audio recording, like in focus groups or group model building (Gill et al., 2008; Merriam, 2009; Vennix, 1996), the workshops within the European financial organisation were neither tape recorded, nor transcribed due to confidentiality reasons arising from the security environment of the study. Instead, the assistant/recorder took notes during the workshops. Additionally, after the workshops, the researcher wrote down his memories to enrich the potential insights from the workshops. Generally on the same or the next day, the assistant and the facilitator discussed the notes and added, deleted or adjusted information in case of different perceptions. Such changes were only done if both researchers deemed them to be necessary. In case of disagreement, both opinions were noted down in order to discuss the uncertainties in the next workshop or with content experts.

The parts below illustrate the discussed, refined, and if necessary double checked notes of the group model building workshops. As indicated above, the causal diagrams from the group model building workshops represent the major data source. Notes from the workshop, conversations, unstructured interviews or observations were used to round off the findings. Since the notes were taken to enrich the causal diagrams from the workshops, the notes do not have the form of a full text as in interviews and conversation.

II. C. 1 Notes on Group Model Building Session 1, 16 March 2017

One of the concerns refers to “specialised islands”. Departments only know about their own department and not about other departments which leads to some kind of silo thinking.

Specialised Islands (Rival Theories)

Note: Participatory modelling and causal diagrams may help to overcome this problem.

Theoretically, the Backlog is related to all of the stocks in software development and operations. It is the pending jobs. The Backlog is considered as a continuous flow of development of software.

Backlog (Software Development)

Items on the Backlog are prioritised.

Agile, (Software Development), Self-Organisation (Pressure)

DevOps decide how many items from the Backlog they can manage (according to their capacity). Basically, Software under Development pulls items from the Backlog and the Backlog does not push items in.

Agile, (Software Development), Self-Organisation, Push & Pull (Pressure)

***Note**: Minimum Viable Product and pulling like in Lean Management and Kanban. Only the first sprint functions by push.*

A sprint is between two to four weeks long. It is a cycle that is repeated to add more functionalities and to improve the software based on customer feedback. Additionally, errors are fixed with new cycles.

Agile, (Software Development), Business Orientation (Pressure), Error (Defects & Vulnerabilities)

Testing and Development are both part of the activities within a sprint.

Software Development, Agile, (Software Development)

If work pressure is high, it is possible to bring more teams to address the pressure. In other words, when necessary it is possible to get people.

Work Pressure (Pressure), Increase Team Size (Security Measures)

The agile approach clearly says that there is no pressure!

Agile (Software Development), Pressure (Pressure)

***Note**: Quote 1*

The software is developed in small parts which are brought to production [release] and which are improved later in further cycles. Particularly in agile, software is developed for functionality because of the customer-oriented approach and security is then seen as add-on.

Agile (Software Development)

Agile, Flexibility, Adaptation (Software Development), Trade-Off Functionality and Security, Business Orientation (Pressure)

***Note**: Quote 13*

The idea of agile is the cycle (see causal diagrams).

Agile (Software Development)

Note: The model should run in sprints (Weeks).

Prior to production, the software is checked to know whether enough has been done to release the feature. The software has to be tested by the DevOps and pentested by others before it is brought to release.

Agile, Secure Software (Software Development)

There are also regular pentests every X months which determine the need to change software. These tests are also considered in the Backlog if they are announced. If they are not announced, they are not on the Backlog for the DevOps.

Secure Software, Backlog (Software Development), Security Approaches

When errors are found in tests prior to release they are generally also fixed before release.

Note*: Important comment because it indicates that fixing defects prior to release does not mean that there is a problem, but simply that errors are resolved before software goes to production.*

Error, Fix (Defects & Vulnerabilities)

Flaws and Bugs may be measured by the "maturity of a team" because the more mature a team is, the less mistakes the team members do.

Flaw, Bug, Mistake (Defects & Vulnerabilities), Maturity (Maturity & Training)

Note: You could correlate the maturity of the team and the amount of incidents to measure the number of errors. Interestingly, this is not done yet.

Maturity (Maturity & Training), Errors (Defects & Vulnerabilities), cyber attacks (Adversarial Behaviour and Cyber Attacks)

Maturity is influenced by practicing the same work and product or changing it, by changes in the team composition (relocation, restructuring, layoffs, hiring), and by education. Education increases the maturity because the more you learn and know, the more you can work on and cover.

Maturity, Training, Restructuring (Maturity & Training),

When you give education to the DevOps teams you decrease the "velocity" of the team because they go to training instead of work.

Note*: Similar to the mechanism described in process improvement*

Velocity, Disruption (Software Development), Pressure (Pressure), DevOps, Training, Process Improvement (Maturity & Training), *(e.g. Sterman, & Repenning, 2002).*

At the same time, increasing maturity is important because we have higher productivity and more security awareness and less mistakes.
***Note**: Quote 32*

Software Development, Productivity, (Software Development), Mistakes (Defects & Vulnerabilities), Maturity, Training, Awareness, Security Awareness (Maturity & Training)

Third Party Software is not always tested, that depends on the case.

Third Party Software (Rival Theories)

Bad libraries: You can use the wrong library because you do not have enough experience. You can also use the wrong library because you have more experience because you use libraries more, so the chance of a mistake goes up.
Note: While not considered here because technical aspect, very interesting and counterintuitive finding. Since organisations rely more and more on libraries due to their efficiency, this may be interesting for future research.

Technical Problem, Technical Solution (Technology)

Standardised Solutions & Technology (Future Research)

The most expensive place to find an error is in production, and when that happens it gets high priority.
***Note**: Very important, matches with literature, e.g. Boehm, 1984 or Stecklein et al., 2004.*

Costs, Financial Risk, Stress (Pressure)

Known errors in a sprint sometimes become unknown errors after release because the DevOps do not tell it to the testers, so that the testers do not only focus on those errors.
Note: There was ambiguity whether this is actually the case. No further information confirmed this idea.

Error, Software Vulnerability (Defects & Vulnerabilities); Employee Indication of Problem: Vulnerabilities due to Miscommunication (Aspects of Perception of Management and Employees)

If there is a severe error, then the software is not released. It is better to test and have low productivity during the sprint, but to

See next page.

have a higher productivity later.

Note: *Interesting insight because this was confirmed over and over again during the case study. Worse-Before-Better-Effect.*

Agile, Software Development, Productivity, (Software Development), Test (Defects & Vulnerabilities),

Try to solve dependencies before putting something on the Backlog. Dependencies are considered to lead to lower productivity and should not exist anymore.

Note: As the topic of dependencies has arisen over and over again, it is definitively an example of a another explanation for issues in software development.

Dependencies (Rival Theories)

Tests come from DevOps tools.

Test (Defects & Vulnerabilities), Technical Solution (Technology)

Tests are done because they are stated in our way of working.

Note*: Compliance culture. The more standards there are, the more work there is. Hence, standards create work.*

Test (Defects & Vulnerabilities), Standards (Security Measures) Overregulation (Rival Theories)

Tests started because there were incidents, so testing tools were created.

Note: Interesting feedback loop.

Test (Defects & Vulnerabilities)

Tests often come from best practice.

Note: Outside the boundaries.

Test (Defects & Vulnerabilities)

A problem for productivity is that many tests are done more than once by different teams. That decreases the productivity of the whole company, but not the productivity of the team that os performing the test.

Note: Could be seen as a “Fraction of Inefficiency.

Note: Policy: Align workflows.

Productivity (Software Development), Dependencies (Rival Theories)

Improvement (Future Research)

Tests are generally automated.

Test (Defects & Vulnerabilities), Technical Solution (Technology & Innovation)

Outside pressure makes you try to do more.

Market Pressure (Pressure)

***Note**: Market Pressure / Pressure from Competition*

External stakeholders (product owner, higher management) create pressure. The product owner is often part of the team, then it is outside pressure that makes the product owner push the DevOps.

Stress, Pressure, Push, Business Orientation, Self-Organisation (Pressure)

***Note**: Very important comment: As shown in several other data sources: Business-driven perspective and pressure make product owner push the DevOps.*

Dependency on other teams, e.g. via bad design of software and bad communication between teams. Products should, however, not be designed together. Only one team should be responsible and teams and products should not be dependent on each other.

Dependencies (Rival Theories)

Errors are generally solved in another sprint.

Note: Contradicts to the earlier statement that errors are fixed within the same sprint. Since this was the only occasion were this statement was made with regard to errors (for vulnerabilities this statement is true), this statement was not very much considered.

Error (Defects & Vulnerabilities),

If there is a severe error, two or three people within a team may stop working and move to solve that error. If it is an error it becomes high priority to make the release happen. If it is a vulnerability, it depends on the severity of the vulnerability.

Disruption (Software Development), Error, Vulnerability, Severity (Defects & Vulnerabilities),

Experience and Maturity are covered in the Dreyfus Model.

Maturity (Maturity & Training),

Other DevOps teams do and sometimes test Third Party Software.

Test, Vulnerability (Defects & Vulnerabilities), Third Party Software (Rival Theories)

II. C. 2 Notes on Group Model Building Session 2, 24 March 2017

Review Session 1

Accuracy of a Minimum Viable Product (MVP) is basically equal to the sprint planning. An MVP needs at least one or more sprints.

Sprint Backlog (Software Development)

Critical incidents (meaning critical problems, defects, and vulnerabilities) are solved immediately. This very much disrupts the current Sprint Backlog and decreases productivity. All other issues that are less important are set on the Backlog.

Note: *Business disruption*

Disruption (Software Development); Delay (Pressure); Defect, Vulnerability (Defect & Vulnerabilities); Cyber Attack (Adversarial Behaviour and Cyber Attacks

Tests are part of the regular activities of a sprint. Hence, rename in the model to “tests during sprint”.

Test (Defects & Vulnerabilities)

Some of the participants say that the entire industry is entirely overregulated and use anecdotes to refer to that. Others say that security cannot be overregulated.

In the end, the participants agreed upon saying that overregulation exists once DevOps face the problem of long checklists that prohibit them from actually working. Other forms of regulations (e.g., tests or fixing on time, etc.) were not considered as overregulation. Stated differently, as long as DevOps remain productive, regulation is good and useful.

Productive (Software Development); Regulation (Security Measures); Overreguation (Rival Theories)

A common problem throughout many industries is that within one organisation there are different standards and guidelines all over the world. One of the participants reported an anecdote about another organisation where there have been very different report times for vulnerabilities. Thus, depending on the country, some needed to report within two days, others within months, etc.

Standard (Security Measures)

Important errors that are not fixed immediately [see above] but set on the Backlog for the upcoming sprint still disrupt the Backlog because they really need to be addressed in the next cycle. Productivity drastically drops in those cases.

Productivity, Disruption (Software Development); Errors, Fix (Defects & Vulnerabilities)

Measure quality

Standards (Security Measures)

- You cannot manage what you cannot measure.
- Develop Criteria
- Does everybody have the same definition of quality?
- Less Errors / Features; more Maturity, more Features
- Quality is not about testing but about having less flaws from the start.
- Maturity of processes: Assure quality from the beginning on in upstream activities, not only at the very end in downstream activities. Thus, find defects as early as possible.

Maturity (Maturity & Training)

Technical debt means that what you do not pay now, you pay in the future. It describes procrastination when fixing. At a certain level, the debt has become so high that one is not able to release any features anymore. Thus, the defects have to be fixed first before any development activities ca take place again.

Defect, here technical debt (Defects & Vulnerabilities)

Productivity (Software Development)

Note: Technical debt describes a typical better-before-worse effect.

Due to the self-organising nature of agile software development approaches, DevOps teams decide themselves whether a product is good or bad. The objectivity depends on the team which makes it quite subjective. Hence, generally, there is some kind of self-assessment. There is only rarely external assessment of the quality.

self-organising (Pressure), standards (Security Measures)

Note: Eroding goals vs. pressure.

Vulnerabilities & Adversary Behaviour

The relationship between defects and vulnerabilities is not 1:1. It may be possible though to create average based on past relationships.

Defect, Vulnerability (Defects & Vulnerabilities)

Next to vulnerabilities from internally developed software, there may be unclear vulnerabilities from dependencies, third party software configuration, or software complexity (i.e., none of the defects alone depicts a vulnerability but combined they are one).

Technology (Technology & Innovation); Dependencies, Third Party Software (Rival Theories)

Organisations need to take care of the quality processes of suppliers to ensure own/final quality (e.g., six-sigma).

Often, vulnerabilities reappear after being fixed. Software vendors release a patch to fix the vulnerability and in the next version of the software the old vulnerability is introduced again. *Note 1: While third party software is only addressed to a limited extent in this study, this mechanism may be a very interesting one to study.*

***Note 2**: Zero-Day Vulnerabilities are a regular consequence of such behaviour.*

Vulnerabilities, Zero Days (Defects & Vulnerabilities); Third Party Software (Rival Theories)

An adversary needs often more than one vulnerability to breach an organisation's system.

Vulnerabilities (Defects & Vulnerabilities); Cyber Adversary (Adversarial Behaviour and Cyber Attacks)

Many attackers use automated tools. After one successful attack they run automated attacks against many other targets, attempting to exploit the same vulnerability. It is not that an attacker desperately hopes for success with one target but rather follows a run, wait, and see approach. Such automated solutions are the same vulnerability scanners (e.g., Metasploit) as an organisation may use.

Attacker (Adversarial Behaviour and Cyber Attacks); Automated Solution (Technology & Innovation)

Some of the participants suggested that no motivation for is necessary for such attacks. Other did not really agree with this. In a conversation after the session, both facilitators agreed that at least some minimum level of motivation for an attack (not for an attack against a specific target) is necessary because else there would be no attack at all.

Adversary Motivation, Effectiveness (Adversarial Behaviour and Cyber Attacks)

Attackers share information and knowledge with each other. Sharing knowledge means often sharing workload as knowledge is the major issue for a successful hack. Hence, attackers increase their productivity by sharing knowledge.

Adversary Strategy; Effectiveness (Adversarial Behaviour and Cyber Attacks)

Automation is not necessary for the success of an attack. A mature attacker may be successful without automation. Automation makes it easier though.

Attack (Adversarial Behaviour and Cyber Attacks); Automated Solution (Technology & Innovation)

While automated attacks have a very low effectiveness as many attacks are blocked, the human productivity (which may be seen as the bottleneck) is improved due to automation because humans only need to look at those attacks that are actually promising.

Effectiveness; Adversary Strategy (Adversarial Behaviour and Cyber Attacks); Automated Solution (Technology & Innovation)

It is possible to run fully automated attacks.

Attack (Adversarial Behaviour and Cyber Attacks); Automated Solution (Technology & Innovation)

Changing or staying with a target does not follow a simple mechanism. Attackers who are not interested in a specific target may lose motivation to attack that specific target again, and thus, change after an unsuccessful attack. Attackers who are interested in a specific target or who have gathered many information about a target are less likely to change after an unsuccessful attack. Particularly attackers who try to hack into a system need specific and good information about it. In other words, some hackers really need to know their target for being

Adversary Strategy, Target (Adversarial Behaviour and Cyber Attacks) successful, thus, changes are vey unlikely for them, no matter the outcome of an attack. Other attackers who do not chose a target but attack randomly change after an unsuccessful attack.

Vulnerabilities are categorised by their severity / criticality: Critical, high, medium, and low.
Note: For modelling you can use either a randomiser, or calculated general fractions (data sets may be publicly available).

Vulnerability, criticality (Defects & Vulnerabilities)

We need to have a very short mean time to resolve to reduce the risk.
***Note**: Quote 26*

Mean Time to Resolve (Defects & Vulnerabilities)

Technology describes an exponential growth pattern and topics include blockchains, artificial intelligence, quantum computers, etc. Those aspects may change the entire game of ICT and cyber security.

Technology, Innovation, Change in Future (Technology & Innovation)

The overall numbers of attacks against the financial industry is decreasing. Nowadays organisations like Amazon are becoming more and more a target because they are less protected and regulated.
Note: This statement seems to be a believe and less a fact as literature and reports document the contrary: Financial organisations are still prime targets for cyber attacks.

The footprint of an organisation is very important to determine the real chance of being attacked.

Take care of Footprint -> security measure

The effectiveness of an adversary is determined by his/her motivation, maturity, and resources. An adversary always need motivation to attack but resources and / or maturity are less important. Thus: Motivation * (Maturity + Resources). Determining

Effectiveness (Adversarial Behaviour and Cyber Attacks) the effectiveness of an attacker in this way makes him/her very similar to an organisation. In this sense, attackers are not different from companies, they have objectives, teams, maturity, tools, and so on. They are just using different, and illegal methods.

Business Case, Adversary Strategy (Adversarial Behaviour and Cyber Attacks)

***Note**: Quote 29*

Once an adversary is successful he/she escalates and attacks again, and again, and again. Such further attacks may represent an Advanced Persistent Threat (APT) as the attacker may dive deeper into the target's ICT system with each successful attack. It may be that only successful prevention and attack mitigation may stop this reinforcing feedback loop.

Escalate (Adversarial Behaviour and Cyber Attacks)

Security Approaches, Response (Security Measures)

If an adversary gets detected while being in the system an organisational response is started. It may, however, be that the adversary is too fast and has already left the system again.

Note: Either stealthy under the radar or fast and heavy in but also fast out.

Response (Security Measures)

Organisations do not always kick an attacker out. Instead, they inform a government which can arrest the attacker.

Note: The private organisation itself has no mechanism to arrest the attacker.

Forensic investigations decrease the productivity of a sprint because DevOps need to be involved when scrutinising the causes and consequences of an attack.

Productivity (Software Development)

Responsible Disclosure

White hat hackers are neither hired, nor does an organisation have a contract with them. There is just a collaboration based on official regulations.

White hat hackers leave if there are no vulnerability findings, not enough payment, a mismatch between the expected and the actual payment, a payment that takes too much time, or an organisation that simply does not fix the reported vulnerabilities.

Responsible Disclosure (Security Measures)

Hackers (both malicious and ethical) are attracted when a well-known company releases a new product (e.g., service, website, etc.).

Responsible Disclosure (Security Measures); Attack (Adversarial Behaviour and Cyber Attacks)

Organisations generally do not pay for internally known vulnerabilities. Since this is demotivating for an ethical hacker, organisations need to fix vulnerabilities fast, so that known vulnerabilities are not reported within responsible disclosure.

Mean time to resolve (Defects & Vulnerabilities); Responsible Disclosure (Security Measures)

Holiday seasons describe the best time to attack any organisation as less people are available for detection and response. On Christmas Eve the number of cyber attacks drastically increased in the last years.

Attacks, Adversary Strategy (Adversarial Behaviour and Cyber Attacks)

Organisations do not invite ethical hackers to conduct responsible disclosure activities.

Responsible Disclosure (Security Measures)

If an ethical hacker does not only conduct responsible disclosure but actually misuses the vulnerability and reports afterwards it is a crime which organisations generally report. This situation is very dangerous for an organisation though as suing an “ethical” hacker may spread in the community which could cause many to stop collaborating with the organisation.

Responsible Disclosure (Security Measures); Attack (Adversarial Behaviour and Cyber Attacks)

Communication and trust are very important in responsible disclosure! If expectations are mismatched and trust lost, ethical hackers may turn to malicious hackers.

Responsible Disclosure (Security Measures); Escalation (Adversarial Behaviour and Cyber Attacks)

II. C. 3 Notes on Group Model Building Session 3, 28 March 2017

Review Session 2

Attacks are not always for getting money but for getting resources. Hence, change "Adversary Budget" to "Adversary Resources".

Resources (Adversarial Behaviour and Cyber Attacks)

Organisations learn from the adversary and from other organisations (external threat intelligence and security cooperation)

Learning (Maturity and Training); Escalation, Arms Race (Adversarial Behaviour and Cyber Attacks)

In the end, it is about the comparison between companies: If other companies pay more then ethical hackers might leave and go to those other organisations.

Responsible Disclosure (Security Measures)

The meantime to resolve affects the trust of the ethical hacker towards the organisation he/she is collaborating with.

Mean Time to Resolve (Defects & Vulnerabilities) ; Responsible Disclosure (Security Measures)

Measure quality

- Incidents / Findings
- Incidents / Tests
- True / False
- Known Vulnerabilities not solved yet
- DevOps maturity as it indirectly increases due to incidents
- Responsible Disclosure Incidents / Reports
- Surveys with ethical hackers
- Threats vs. Controls
- Risk and Threats: Risk is the chance a threat occurs.

Standards (Security Measures)

Threats and vulnerabilities may remain the same but the risk can change. This depends on the risk appetite: 1. Accept the risk, 2. Avoid the action, 3. Mitigate the impact and the occurrence, 4. Insure against the risk.

Financials (Future Research)

Third Party Software (TPS)

Third Party Software (Rival Theory) -> Only very interesting / striking aspects are pointed out. Else this is not further coded.

Organisations sometimes collaborate with open source software solutions. Thus, organisations provide support or even DevOps teams for those open software solutions in case of major problems to, for instance, patch it. While this approach exists, it is not common though.

Depending on the contract, some parts of TPS are managed by an organisation and others are done by the software vendor.

Changing or patching, etc. is part of operations/maintenance and thus part of the activities of DevOps. In other words, maintenance is part of the overall work of DevOps.

Workload (Pressure)

Customisation of TPS is part of the work of DevOps as well. Sometimes customising TPS changes a lot and is a lot of work, sometimes it is very little. While customising has the benefit of matching the TPS with an organisation's system, it has the downside that every released patch also needs to be customised.

DevOps and other employees within an organisation are mutually responsible for searching, finding, and assessing new TPS options.

Decommissioning TPS results sometimes in the acquisition of new TPS, but by far not always, particularly not when an organisation wants to have less slack.

Changing TPS and applying the patch to TPS has the same consequences for DevOps regarding their workload.

Workload (Pressure)

Training and Awareness

Awareness for training comes, for instance, via intranet, signposts on the walls within an organisations facilities, best practice, online sources, contact with other teams, directed messages, the maturity within a team because the DevOps talk about issues that make aware, and other aspects.

Awareness for Training, Maturity (Maturity and Training)

Training has an impact on productivity. All time spend in training decreases the productivity of a team. If you overtrain, this productivity loss has quite an impact.

Note: Similar to Repenning, & Sterman, 20002

Productivity, Disruption (Software Development), Training (Maturity and Training)

More mature generally means more aware of training options and of security issues.

Maturity, Awareness, Security Awareness (Maturity and Training)

DevOps become aware by looking at the test results. If those results are discussed within the team over time this increases the maturity of the team.

Tests (Defects and Vulnerabilities); Maturity (Maturity and Training)

DevOps teams and managers plan for training. Thus, it is a regular activity and should not disrupt business.

Disruption (Software Development), Training (Maturity and Training)

You need regular trainings for being aware of potential sources of problems. If you do not train you may not know that there is a problem or what a problem is about.

Security fatigue may arise due to overtraining. Security fatigue is unlikely as long as DevOps do not say “oh no, again a training…”

Overregulation (Rival Theory)

Less than security fatigue, securitisation may lead to mature DevOps become security people which would mean that maturity and security awareness among DevOps decreases.

Securitisation, similar to restructuring (Maturity and Training)

Model Walkthrough

DevOps productivity is affected by incidents, training, and disruptions from defects and much more severe from vulnerabilities.

Productivity, Disruption (Software Development), Defects, Vulnerabilities, Criticality (Defects & Vulnerabilities); Incident (Adversary Behaviour & Cyber Attacks)

The participants agreed with the structure on agile software development, defects, and vulnerabilities.

Vulnerable TPS may cause an incident: If attackers know that a TPS is vulnerable (e.g., a bad library), all organisations employing that TPS may get attacked. At the same time, secure TPS (e.g., good libraries) prevent incidents which attempt to exploit this kind of weakness.

Attack (Adversary Behaviour & Cyber Attacks), Third Party Software (Rival Theories)

Known defects in TPS should increase the number of known vulnerabilities and the probability of being attacked. There was, for instance, an attack exploiting a google vulnerability which afterwards affected washing machines.

Vulnerability (Defects & Vulnerabilities), Third Party Software (Rival Theories)

When you respond to an attacker who is already in, often you are too late. What you do is you try to reduce the impact, learn from it, and go on.
*Note: **Quote 33**.*

Respond (Security Measure)

It is not firefighting when defects are fixed. Then the world has not seen it. But it really is firefighting when the world knows about it, thus when there are known vulnerabilities or known attacks, because those need to be addressed fast.
Note: Very important. There seems to be a line between addressing defects and addressing vulnerabilities and attacks. Follow this further.

Pressure, Firefighting (Pressure), Defects, Vulnerabilities (Defects & Vulnerabilities)

Generally, there are enough DevOps, so some can firefight and others can continue with the regular activities. There would (or should?!) always be a fraction of DevOps that continues working on the regular activities and not go to firefighting vulnerabilities and attacks.

Productivity, Disruption (Software Development), Firefighting (Pressure)

Note: POs may have information on that.

Improving Quality: Policies

Fast feedback loops to the DevOps diminishes the delay between making a mistake and getting to know about it. Teams know the quality of their software while developing. This may be done with automated mechanisms or good standards.

Standards, Measures, System Thinking, Problem Awareness (Security Measures); Automation (Technology & Innovation)

Note: In this way teams would not / are less likely to fall into the adaptation trap because they know the amount of future work.

If you have dependencies you may see the errors of the other teams. Here teams could collaborate and learn from each other. Yet, dependencies also create unknown future workloads as teams do not know how many errors are in a software.

Workload (Pressure), Dependencies (Rival Theories)

Acknowledge security: Train the trainers (and decision makers)

Managerial Opinion, Understanding of Problem (Perception)

Clear assignments, clear concepts: While this decreases the number of misunderstandings and subsequent problems, it leaves little room for creativity which may impede the self-organising nature of DevOps teams.

Self-organising (Pressure), Standards (Security Measures)

Emergency teams, list of people that can help in emergencies. Common in many firms now is let teams fix their own mess.

Awareness, Learning Response (Security Measures)

II. D Documentation Interviews, Conversations, and Observations

Next to the causal diagrams built in (and afterwards based on) the group model building workshops (4; I), and the notes taken during the sessions (II.C), further qualitative (and quantitative) data was gathered throughout the six month on site.[35] As indicated in the methodology-section and in the appendix above, the group model building workshops were followed by further qualitative data collection via e-mail, chat, phone calls, conversations, unstructured interviews, observations, and the review of documents and archival data. The parts below show the notes from unstructured interviews, conversations, and observations. Some conversations or interviews are written down in such a form that they resemble a discussion. This was done in those case were the notes taken by the researcher during the conversation or unstructured interview were rich enough to support such a story-like approach. Yet, since those notes do not equal a tape recording, also the story-like descriptions mainly contain the essence of the interview or conversation. In other cases, for instance, when the researcher was presenting results to the team, there was not enough time to take rich but only more abstract notes. Thus, in such cases only the notes and not full story-like discussions are presented. Observations are always given in form of notes and never in form of a rich story. Notes on documents and archival data are not displayed due to confidentiality. As above, the notes are coded and categories and are sometimes commented.

II. D. 1 Unstructured Interview, 13 February 2017

Interviewer: Hi X, thank you for taking the time to meet with me. Y told me that you would get me a quick start on software development, software security, software vulnerabilities, and cyber attacks. Is that correct?

X: That sounds good, yes we do that. Where should we start? How much do you know about secure software development?

Interviewer: Thanks. I read quite something about the different models that we have out there, like Microsoft SDL or OWASP 10.

Secure Software Development (Software Development)

[35] The quantitative data is not used explicitly in this study for two reasons: First, quantitative data was mainly collected within and outside of the European financial organisation for the purpose of quantifying a mathematical simulation model. As indicated above, no quantitative simulation model was built due to time and data constraints, as well as anomalies within a model which would have been used as basis. Second, due to confidentiality reasons no quantitative data from within the financial organisation can be shown within this study.

I also looked at the one from Adobe and I saw something from literature. What do you think about those process models? Am I missing an important aspect?

X: Yes, you described the well-known approaches. Very good. There is another one that I personally really like because it is theoretical and practical at the same time. It is called BSIMM which means build security in maturity model. It is done by an expert who runs his own company and simultaneously works as a professor at university. His name is Gary McGraw.

Secure Software Development (Software Development)

Interviewer: Yes, I have already stumbled upon the name McGraw. He seems to be one of the main experts in the field of secure software development. Or do I have a wrong impression?

X: No, you are perfectly right. He emphasises the importance of building security in for decades now and he is a well-respected source.

Secure Software Development (Software Development)

Interviewer: X, you talked about the BSIMM and you said that you personally really like it. What is BSIMM about and why do you like it so much.

X: BSIMM is not purely descriptive like many other process models. It is descriptive as an overview and prescriptive as a guideline at the same time. It comes from practitioners who are also active in science, and it is made for other practitioners. McGraw and his colleagues have worked with many partners in different industries and have found out what companies do to stay safe. In fact, BSIMM shows best practice across many industries. It is

Guideline (Security Measures)

based on very rich data from many companies [around 100], compares the different approaches, gives good ideas about pros and cons, and even measures concepts over time.

Interviewer: Great, that sounds very good. I will look into it. Since the BSIMM seems to be so good, is it generally employed in most companies?

X: *Laughing.* Never. Although we have all those models, each company is different. I would even say that in most companies different departments have a different understanding of what a secure software development process is and what should be included. So, no, there is no generally agreed upon framework.

Secure Software Development (Software Development), Employee Problem Understanding / Perception (Perception)

Interviewer: Does that mean that security is not a matter of concern for most or is that rather a communication problem?

Awareness (Maturity & Awareness), Employee Problem Understanding / Perception (Perception)

X: I would say both. Everybody simply understands things slightly different. I think that is normal and part of our nature. But you can also see that there are too little security people in software [the field]. In the BSIMM they [the authors] distinguish between people who belong to the software security group, to the satellite group, and to DevOps. The software security group describes those few people who are experts in both areas. The satellite group is for those people who are expert DevOps and have some knowledge in security. Well, and the DevOps are the rest. You know what DevOps are, right?

Maturity (Maturity & Training)

Interviewer: *Laughing.* Yes, thank’s for asking. Ok, I think I got this. Which other topics do we need to look at? What do you think we should think about when trying to analyse secure software development and cyber attacks?

X: Hm, difficult question. There is lot's to think about… I would say we should discuss the development activities, testing, the problems when you become vulnerable, cyber attacks, and responsible disclosure… Where do we go first?

Interviewer: Since we have already covered the development part a bit, let's go to testing. Why do you test? What do you mean when you talk about testing?

X: Ok, so there are many different tests. Initially, tests were done to see whether a software works how it should work. That's about functionality. In security we have quite some further tests. In software security you test, for example, for the boundaries of your software… that means, does it still function under extreme conditions… then, you test for the requirements… does the software do what it is supposed to do… We also have features testing… you would for example check that an online application has a login and a logout function. You can also do fuzz-testing what is quite new. With that you would for example type in .fra instead of .fr as country code for France. You may not believe but sometimes a successful cyber attack needs nothing more than something like that…

Test (Defects & Vulnerabilities)

Software Security, Secure Software Development (Software Development)

Cyber Attack (Adversary Behaviour & Cyber Attacks)

Interviewer: Wow, very interesting. Since I am entirely new in this field, could you explain how a cyber attack takes place. From the previous project I know how a malware attack works, but how does a hacker get into an organisation's system?

X: *Laughing.* Yes, we can go there. It actually fits quite well because hackers also use all those testing methods. In fact, they use the same tools for checking our system as we do.

Test (Defects & Vulnerabilities), Hacker (Adversary Behaviour & Cyber Attacks), Automation (Technology & Innovation)

There is, for example, the CVE data base which lists publicly known vulnerabilities. So a hacker looks there and then takes a testing tool, for instance Metasploit, and checks a system that he wants to attack.

Interviewer: That sounds quite simple. How does such an attack take place?

X: It can be that simple but it can also be very difficult. That depends on the case, on the target, etc. Anyways, I would say we can summarise a hacking attack in three steps: First, you collect information about your target, such as the web server version, the os [operating system] version, the framework [java, .net, javascript, etc.], or simply browse for weak spots… In a second step, you try to find a vulnerability based on the information you collected. Hackers use automated tools for that or even do that manually if they are really skilled. In a last step, a hacker simply exploits the vulnerability.

Hacking Attack (Adversary Behaviour & Cyber Attacks)

Vulnerability, Weak Spot, Weakness (Defects & Vulnerabilities), Automated solution (Technology & Innovation)

Interviewer: On an abstract level that sounds quite simple.

X: Well, depending on the target, it may be quite simple. Quite often you see numbers in the address field of the browser. You can try to change those numbers and sometimes that is already enough for entering a system because suddenly you have access to something that you should not have access to.

Interviewer: That sounds like even I could hack… *Both laughing…* One thing struck me… You talked about the skills of a hacker. What did you mean with that?

Effectiveness, Hacker (Adversary Behaviour & Cyber Attacks)

X: Hackers have skills sets and many hackers work in teams. So those teams have then specialists for using a specific tool, for knowing which tool or approach to use, for manual hacking, for knowledge on specific systems or aspects like different OS, or even for misusing responsible disclosure programmes… Depending on the case, such teams can be a lose connection between a few script kiddies or it can be a tightly organised team that is part of a larger organisation. Particularly the latter case is then very similar to any normal company. Such criminal organisations can even have departments, different hierarchical levels, duties, and so on… Do you know Z? I think you should talk with him about hacking, he knows much more than I do…

Hacker, Adversary Strategy, Effectiveness, Teams, Learning, Business Case (Adversary Behaviour & Cyber Attacks)

Interviewer: Ok. I see. I will talk with him.

X: I will arrange a meeting for you.

Interviewer: Great, thanks! You talked earlier about vulnerabilities and exploiting those. How can you prevent such a situation?

Prevention (Security Measures)

X: There are many measures in place… For external software that companies buy or rent, you generally see that the software vendor makes a patch available to fix the vulnerability. The problem here is that larger patches may actually cause quite some downtime which would disrupt the business. So, for companies, patching can be very expensive and that can be the reason why it is sometimes not done. The problem there is, if you do not patch such known vulnerabilities, everybody can easily exploit them, except you create other protection mechanisms… For internally developed or customised solutions, you can do pentesting [penetration testing], dynamic analysis which is somewhat a light version of pentesting or responsible disclosure…

Patch (Security Measure), Third Party Software (Rival Theories),

Disruption (Software Development)

Vulnerability (Defects & Vulnerabilities)

Pentesting, etc. (Security Measures)

x

Interviewer: X, we have now covered aspects of software development, software security, software vulnerabilities, and cyber attacks. Is there anything else, you think I should know about?

X: No, I think to start with, we made quite a good tour. Of course, whenever you have questions, come back to me.

Interviewer: Thank you for your time and for helping me getting this project started and for arranging the meeting with Z.

X: No worries, you're welcome!

II. D. 2 Unstructured Interview, 13 February 2017

Interviewer: Hi X, I'm Y. Nice to meeting you.

X: Nice to meeting you too.

Interviewer: Thank you for taking the time to talk with me about hacking. Z told me you're the one I need to talk to when I want to know something about that topic.

X: *Smiling…* Yeah, I know a bit about it… What do you want to know?

Interviewer: Let's start quite simple and direct. What is hacking?

X: *Chuckling…* Quite general, hacking means that you combine things or ideas to do something it was not intended to be used. You can also say, you simply abuse something in an unintended way. In IT hacking means then that you try to find ways to have

Hacking (Adversary Behaviour & Cyber Attacks) more possibilities than the software was intended to provide you with. The problem here is often that requirements are not well set. The desired software behaviour for people from operations and business is not security. They do not care because security doesn't give money and if nothing happens nobody even recognises the success of security because you don't know why nothing happened. **[Quote 14].** So, that is quite dangerous because hacking is about thinking about ways how to go around or misuse an IT solution. It is easier as an attacker to misuse one mistake than to prevent all mistakes as a developer. And many developers are also simply not trained to think like a malicious hacker. They do not develop software an think at each place about how to misuse something.

Business Risk, Security Risk, Costs (Pressure)

Attacker Advantage (Adversary Behaviour & Cyber Attacks)

Lack of Training (Maturity & Training)

Interviewer: Thank you, that was quite a nice introduction to hacking. Let's make on step further and come to hackers. What is important for hackers?

X: Hackers and attackers in general are in a better position than defenders. They are not time bound, they have plenty of targets, and they only need to find the weakest link. Programmers, in contrast, need to think about all problems that can exist. And the attack surface of a global organisation is really large. Think about all the employees that are spread over so many countries. I would even say that every public asset of every organisation is constantly under attack.

Attacker Advantage (Adversary Behaviour & Cyber Attacks)

Interviewer: That does not sound really promising…

X: It's also not that bad. *Laughing…*

Interviewer: *Laughing…* Well, if you say so, I trust you with that.

You're the expert. X, we've talked now about what hacking is and about important aspects for a hacker. What I just realise, we have not come to the topic of how a hacker works. I mean… *chuckling…* in movies you always see guys with hoodies who type something on a keyboard… is it really like this?

X: *Laughing…* it depends. But yes, it can be like that… I would say hackers normally gather information about their target, they try to find a way in, and they test if that way actually works.

Interviewer: So, you mean, first a hacker searches for information, then he tries to find a vulnerability, and then he tries to exploit it… is that correct?

Hacking (Adversary Behaviour & Cyber Attacks)

X: Yes, indeed. In the first step you would try to find out what a target's business is about, where they are located, what kind of people work for the company, what kind of websites there are, etc. You have many options there, like social media, company homepage, job descriptions, etc… If you find something that looks older then it is likely that it is more vulnerable…

Vulnerable (Defects & Vulnerabilities)

Interviewer: Ok, that was the first step. How is it in the next one?

X: The easiest way in is via a combination of social engineering and malware. For me that is still part of hacking. You just convince somebody to click on a link or download the attachment and then you have your malware on the other's asset. Or you manage to even get things directly done like with the CEO fraught… You heard about that?

Hacking with malware and social engineering attacks (Adversary Behaviour & Cyber Attacks)

Interviewer: No, not really… should I?

X: Ah, no worries… It's quite simple… basically you call the secretary of the CEO and tell that person that the CEO told you to call to get a super important transfer done. It's a minor thing, just some 250.000$ or something… and if the secretary hesitates then you point out that you can totally understand that but the business will not happen and then the secretary has to explain that to the CEO and live with the consequences… it has been done quite a lot…

Interviewer: That is quite a tricky way of getting money.

X: It is… So, in any case, you try to find a way in, no matter how you do it. And then you exploit the vulnerability. Generally, you exploit a vulnerability in one of the public facing assets [those are generally websites, employees, physical facilities]. And the exploitation is quite simple. You try negative and positive numbers in a field, you change the numbers in the address field in your browsers, you check about rounding errors, or you search for logic errors. In the end, it's always about those cases that are on the edge.

Vulnerability, Public Facing Asset (Defects & Vulnerabilities)

Interviewer: So the exploitation is also the last act in the system or does the hacker dig deeper?

X: That depends. Those cases above are quite simple but can be done over a longer time as long as the bug is not fixed. If you have a more sophisticated attacker, then that one can stay in your system and try to increase his privilege level to be able to do more stuff. Or the guys can move in your system and search for vulnerabilities in non-public facing assets.

Bug, Vulnerability (Defect & Vulnerability), Attacker Effectiveness, APT (Adversary Behaviour & Cyber Attacks)

Interviewer: Ok, I guess, I start to understand this more and more… how can an organisation prevent such attacks?

X: The configuration of your systems is quite important. A web server should, for example, have a low privilege level because it does not need more and because it is dangerous if it has more. And in addition, companies have prevention, detection, and response measures… If you only prevent you will lose because there is always a weakest link… So, you have a problem when an attacker is inside your system because you do not recognise that without detection.

Prevention, Detection, Response (Security Measures), Adversary Advantage (Adversary Behaviour & Cyber Attacks), Technology, Configuration (Technology & Innovation)

Interviewer: How does such detection work?

X: Without going into details, it can be quite simple: If a computer tries to connect to the entire network or at a very strange time of the day, then I would say most systems see that as suspicious… And if you have something like that, you could get the guys from CERT and so on to respond. Nowadays most systems have security in depth which means that you have security measures on each layer of the system… for example… your software is up to date, your system has the most secure configurations, your people are well trained and so on… Of course, you have situations where you cannot do much. If your OS has a huge vulnerability that you do not know about because nobody, not even the vendors knows about, then you have a problem… I mean, everybody who uses that OS has then a problem…

Detection, Response, Cyber Resilience (Security Measures), Configuration (Technology & innovation)

Zero Days (Defects & Vulnerabilities)

Interviewer: That would be a zero-day if I'm not mistaken?

X: That is correct…

Interviewer: To find such a zero-day… how do you do that? Do you need to be lucky? Or is that skill?

X: Hacking is about creativity. Creativity is the most important skill of a hacker. Attackers have always somebody in their team who is creative. Hackers often work in large teams, especially when we talk about criminal organisations or governmental forces. Those groups work like companies, they have managers, business people, perhaps even people from legal… and the hackers are specialised in many different areas, like physical security, operation systems, webservers, networks, software applications, automated tools, brute force, cyber fraud, zero days, phones, and so on… Many hackers use automated tools because it makes it easier but you really don't need that for hacking… There is for example this guy on Youtube who breaks into the physical facilities of a bank with whisky…

Hacking, Skills, Effectiveness, Attacker Maturity (Adversary Behaviour & Cyber Attacks)

Adversary Strategy, Business Case (Adversary Behaviour & Cyber Attacks)

Automation (Technology & Innovation)

Note: Of course, the interviewer and X watched the video.

Interviewer: Unbelievable… I really see that it is about creativity… X, you sad earlier that you respond on an attack, when somebody is in. How does that happen.

X: Very general, you try to find the root cause of the incident and you try to get the guy out.

Disruption (Software Development), Response (Security Measures), Attack (Adversary Behaviour & Cyber Attacks)

Interviewer: Is that possible?

X: Generally yes, but it can be really hard. Battles between hackers and defenders can go over several months. Often you think the guy is out but then he was just stealthy or had a backdoor or something else and is still inside. If you have, for example, an attacker with a high privilege level, you really

Time of an Attack, Disruption (Software Development), Pressure (Pressure), Hackers, Attack (Adversary Behaviour & Cyber Attacks) have a problem because once you think that you have him out, he is just somewhere else because he could give himself the rights to be there… I remember, there was this case of a smaller company where the people went back to work and the attacker was still inside because they could not afford it anymore to neglect their normal jobs. At the end of the day, they needed some kind of external security provider to get the attacker out…

Note: ***Quote 36*** *[parts in the middle are not used].*

Interviewer: That sounds quite heavy. Was that because the company was so small or could that happen with anybody?

X: I guess the size played a role but theoretically speaking, this could happen to any organisation…

Major Disruption possible (Software Development)

Interviewer: Ok, I see. X, thank you very much for all those insights. Do you have anything else you want to share with me? Otherwise, I do not want to steal more of your time! *Chuckling…*

X: Ok… If you have any further questions, just come over.

Interviewer: Thank you very much. Have a nice day.

X: You too.

II. D. 3 Observation, 27 February 2017

The following quotation was written down during a meeting with several members from the financial organisation. The meeting was not set for the purpose of this study.

X: We are a business, we have no unnecessary slack in this organisation and people are not just sitting around and wait to do something! [**Quote 3**]

Emphasis on Business (Pressure)

II. D. 4 Unstructured Interview, 17 March 2017

This meeting took place with several colleagues from the European financial organisation. The researcher joined the meeting for two reasons: First, the meeting was an interesting learning opportunity, second, he could ask questions on vulnerabilities. In the light of this setting, the notes below do not represent all aspects discussed during the meeting but only those which are interesting for this study and which are at the same time not confidential.

[…]

Interviewer: So what do you mean when we talk about risk, and threats, vulnerabilities, and all those things. In other words, where is the difference between the different aspects?

X: You take the vulnerabilities and the threats and you have your risk. You take the risk and the measures that are in place and you can talk about a successful or unsuccessful attack.

[…]

X: So you have critical, high, medium, and low vulnerabilities. Software vendors provide patches when there is a vulnerability but this may quite some time. So the point in time when you detect a vulnerability is not the same when it gets fixed. When you scan for vulnerabilities you should really not find the same vulnerability twice in a row because the vulnerability should have been closed already…

Fix, Criticality, Mean time to resolve (Defects & Vulnerability), patch (security measures), Third Party Software (Rival Theories)

[…]

Interviewer: What is the impact of vulnerabilities - no matter whether they are internally developed or externally supplied - when one finds them? What does that mean for an organisation?

X: Critical vulnerabilities disrupt the sprint backlog of a DevOps team. They get the information, they change their priorities and they deal with it. Other stuff just gets on the backlog.

Disruption (Software Development)

[…]

Interviewer: How does vulnerability detection work?

Test / Detection (Defects & Vulnerabilities)

X: There are many options… Vulnerability scanners, for example, can only find known vulnerabilities. So if you know about a vulnerability due to a vulnerability scanner, then anybody else may know about that vulnerability too because they may use the same scanner. If you rely on pentesting, dynamic analysis, red teaming or responsible disclosure, you may find unknown vulnerabilities. All of those approaches have their limitations though. For example, responsible disclosure is pretty limited because it is only for web applications and public facing assets but for nothing that you have internal. And in addition you could question whether it makes sense to pay for the findings that are about all those preventable low hanging fruits…

Several security measures (Security Measures)

[…]

Interviewer: So far we talked about many different aspects related to vulnerabilities. Would you mind if we turn to internally developed software?

X: No, perfectly fine for me… but what do you want to know?

Interviewer: Across all the different industries there are all these rules, these standards, these measures, and so on, and yet there are really avoidable software vulnerabilities that are even listed in OWASP Top 10. What do you think, why is that the case?

X: There are many reasons. The quality of people… many developers have simply no idea about security. Then, there is a clear lack of discipline… I mean, people know about rules and in many cases, people still do not care. You see that particularly in the industries where fast delivery counts… Yes, so there is time pressure. And then there is business. The product owner from the business side looks first at functionality because that creates income, then at security because that costs money. We have enforceable standards in place, so people need to consider both, functionality and security. When there are deadlines this puts them under stress.

Reasons for software vulnerabilities

Maturity, Security Skills (Maturity & Awareness), Lack of Compliance (Security Measures)

Time Pressure, Stress, Business Focus (Pressure)

Note: ***Quote 2****.*

[…]

II. D. 5 Conversation, 20 March 2017

Interviewer: Thank you very much for taking the time to shortly have a chat. I would like to describe you the following theoretical situation and would be happy to hear your thoughts about it. Would you be ok with that?

X: Yes, of course. Go ahead.

Interviewer: A lot of people in security are creating new rules, regulations, standards, etc. to improve security. This certainly has a good effect and decreases the amount of avoidable problems.

Rules, Standards, etc. (Security Measures)

I'm wondering though whether it could be possible that from a certain point on there are so many regulations that the quality of work actually suffers from the amount of regulations that need to be taken into account?

Overregulation (Rival Theory)

X: Good question! Well, I would say, for now we certainly do not have it because we are just on the way of normalising and organising the different areas.

Interviewer: Ok, I understand, so for now it is fine. Sorry for probing on this, but do you think that overregulation could become a problem in the future?

X: No worries. You're right, I very much see your point that there is a certain danger that we actually make things worse by trying to make them better. It is true that it is certainly not the amount of rules we create that make things better, but the quality of our work.

Systems Thinking (Security Measures)

Interviewer: I see. So, to sum up, for now the organisation is doing well, but it could be that some kind of overregulation arises in the future.

X: Yes, that is very correct and I think that is a very interesting thought. Thank you for bringing this up.

Interviewer: Great, nice to be of help and thank you very much for your time and your responses. Have a nice day!

X: You too.

II. D. 6 Unstructured Interview & Validation 1, 3 April 2017

This unstructured interview was conducted in order to validate the outcomes of the group model building session. As common, the researcher used the approach of disconfirmatory interviews for validating the model and the results. As described in II. A and II. E., the model was shown to the expert by unfolding it feedback loop per feedback loop. The researcher presented it and probed the expert each time to criticise, change, adjust, and disconfirm the model. Due to the nature of this meeting, the text below does not show the entire interview. Instead, it provides the introduction, and notes about the comments from the expert, and responses from the researcher on the comments. The expert took part in the first group model building session and is familiar with system dynamics terminology and methodology.

Interviewer: Hi X, thank you for taking the time to meeting Y [the gatekeeper] and me today. Considering your busy schedule, I very much appreciate that.

X: That's alright. It’s fun working with you and I like the modelling, so it's good for me.

[Nice chat, arranging room together for using the screen, etc.]

Interviewer: Ok, I think we can get started. Everything good with the two of you?

X: Yes, let’s start.

Y: Fine for me as well.

Interviewer: Ok. X, I am going to present the entire model to you. Since the model is pretty large by now - your colleagues made a good job there - I will unfold it basically one feedback loop after another. What I want you to do is the following: Try as hard as you can to criticise and disconfirm what I am explaining to you. I would like you to challenge me, so that I really need to explain well what we have done here. I want you to do that because I want to make sure that we actually have a good product at the end.

X: Ok, I can do that… *Laughing…*

Interviewer: Great. I would like to ask you to look at the model particularly from the following three perspectives: Structure, Variables, Boundary [the researcher wrote those three aspects on the white board]. What I mean by that is: Does the chosen structure represent the real system? Do the chosen variables exist in reality? And have we drawn the right boundary or are we excluding something we should not exclude or do we include something that is not necessary to include because it does not add value. You remember for example from the first workshop that we said that we exclude technology in the sense that we do not model technical details. That is for instance a boundary we drew. Everything clear?

X: Yes, perfect. I guess we can do that.

Interviewer: Great, let's start. And please, always think about telling me what we did wrong here. Tear this model apart if necessary, but tell me what needs to be changed.

X: *Loud laughing…* I won't let it survive! *Laughing…* Go ahead.

[…]

X: You're right the amount of regulations depends on attacks and best practice but it also depends on the DevOps team. A mature DevOps team needs little regulations and is not really affected by it because it follows the underlying ideas of the regulations anyways due to its maturity.

Maturity (Maturity & Training), Regulations (Security Measures), Attacks (Adversary Behaviour & Cyber Attacks)

Interviewer: So, the more mature a team, the less regulations it needs.

X: Yes, that it correct.

Interviewer: But the regulations still exist?

X: Yes.

Interviewer: So, the number of regulations remains the same but the effect on productivity within a mature team decreases. Is that correct?

Productivity (Software Development), Maturity (Maturity & Training)

X: You could say so, I believe.

Interviewer: Are all DevOps team equally mature or are there quite some differences?

X: *Grim smile…* There are obviously quite some differences. That is normal with people. Why are you asking?

Interviewer: Well, then I would suggest that we keep this connection in mind but do not draw it because the benefit from mature teams is offset by the immature teams. What do you think?

X: Hm… *mumbling…* Yes, I believe that is correct. Sounds good.

[…]

X: The productivity of a DevOps team depends on its maturity. There is knowledge on the distribution of maturity amongst and within DevOps teams in the organisation and there is knowledge on the productivity of different maturity levels. You should check that.

Productivity (Software Development), Maturity (Maturity & Training)

-> E.g. Dreyfus

Interviewer: Thank you, I will.

[…]

X: Ah… I disagree there… The problem is less that we have a securitisation and turn all our good people in DevOps to security people. This has probably only very limited impact. So, one problem we see is that mature people are more likely to leave to other companies. So when we train DevOps, we need to consider that they may leave. I believe training and afterwards keeping mature people is the key.

Securitisation, Mature Leave, Layoffs (Maturity & Training)

Note: Important insight. Disagreed.

Interviewer: How does this occur? Could you elaborate on that.

X: Of course people do not leave immediately when they reach a certain level of maturity, it is rather a delayed effect, and it only starts from a certain level of maturity. But I would say, it significantly influences us and I believe any other organisation out there. The challenge is really to keep these people.

Keep maturity (Security Measure)

[…]

X: I see little relevance for including third party software. Why do we include it here?

TPS (Rival Theory)

Interviewer: Being busy with third party software takes time from the DevOps because they need to choose what to get, they need to test the software, they may customise it, and they need to communicate with the respective contract partner so that the problem gets solved. In addition, errors or flaws in third party software may lead to vulnerabilities. What do you think about it. You did not seem to be persuaded, shall I delete this part of the model?

Productivity (Software Development), Workload (Pressure), Vulnerability (Defects & Vulnerabilities)

X: No, not at all. It makes sense that the DevOps teams are busy with third party software. But if we go into detail, I would say that the most time consuming task here is the analysis of test results and of found errors.

Interviewer: Ok, I understand. So with third party software it really depends on the task.

X: Yes, that’s right.

Interviewer: Is there anything else you would like to point out with regard to third party software? Anything that disturbs you in the model? Anything we’ve missed?

X: Errors in third party software could hinder internal software to work. If you have dependencies, or if something from our stuff would become vulnerable because of the external solution, then you have an impact there.

Dependencies, TPS (Rival Theory)

Interviewer: I see. In sum, third party software is relevant but it depends on the tasks whether it actually takes time.

X: Yes, that is true.

Interviewer: Nothing else to change?

X: *Laughing…* no, looks fine.

[…]

X: Tests do not take that much time anymore. Like, you know, they have really only little impact on the productivity of DevOps because of automation. What still is an issue is the distinction between true and false positives because that can take quite some time. But again, like so many other things, that depends on the maturity. More mature teams are much better here. Actually, true and false positives is even one common criterion for assessing the maturity of a DevOps team.

Productivity (Software Development), Automation (Technology & Innovation)

Maturity in testing (Maturity & Training)

Interviewer: Ok, so, tests have generally limited impact on the productivity of a team. If a team and its tools are mature this is even less the case. If they are less mature, then productivity is more affected. Is that what you said?

X: Yes, pretty much. Generally, tests are really not a major issue anymore when we look at sprints and productivity.

Productivity (Software Development), Automation (Technology & Innovation)

Interviewer: Ok. Got it.

[…]

X: Yeah, I would say that's correct. You can generally say that known errors also become known vulnerabilities. Some of the errors may not be perceived as vulnerability and then you don't know it but that is rare, I would say. Where I am not sure is how you see the relationship between errors and vulnerabilities. Do you see that as 1:1?

Error - Vulnerability - Relationship (Defects & Vulnerabilities)

Interviewer: No, not at all. You and the others have emphasised quite a lot that such a clear ratio does not exist, at best we may have an average.

X: Yes, that sounds correct. Ok.

[…]

X: Stop there, I think that is wrong. DevOps have nothing to do with the detection of vulnerabilities. So detection of vulnerabilities should not be connected to DevOps productivity.

Productivity (Software Development) Vulnerability Detection (Defects & Vulnerabilities)

Interviewer: Ok, interesting. Then we have obviously missed that link throughout all those sessions. Do you want me to delete it?

X: Yes, I would say so. The DevOps really do not do the vulnerability detection.

Interviewer: Ok, I'll take it out.

[…]

You always resolve the critical vulnerabilities immediately. You cannot put them on the Backlog. The rest, that's a different thing… In many industries vulnerabilities are actually not fixed on time. But you have to see that vulnerabilities can be exploited immediately, so you need to fix fast!

Disruption, Backlog (Software Development), Pressure (Pressure), Vulnerabilities, fix fast (Defects & Vulnerabilities)

Note: ***Quote 25***

Interviewer: Ok, so I will emphasise here the need to reduce the mean time to resolve?

X: Yes, that's it. We need to find and fix fast.

[…]

X: Hm…

Interviewer: So we found something. What do you think? What do we need to change with this mechanism?

X: I would say that this one there is only half of the story… Generally, an attacker changes his target after an unsuccessful attack. If the attacker has a specific objective and conducts a targeted attack against an organisation, he will stay with that target because he has more information about it and he has a reason to attack that target.

Adversary Strategy (Adversary Behaviour & Cyber Attacks)

Note: ***Quote 31***

Interviewer: Ok, so I will add that part after the meeting?

X: I would say yes. At the same time, an attacker may change the target if he wants to exploit another target with the same vulnerability. The exploited company knows now about it but since many companies do not admit that they were successfully attacked, an attacker can just go the next and do the same trick there again…

Adversary Strategy,Business Case (Adversary Behaviour & Cyber Attacks)

Interviewer: Ok, I'll add that too?

X: Hm… No, I think you don't have to. I think the first mechanism is more important… this one may be too detailed…

Interviewer: Ok, so I will add the first one so that we show that an attacker can stay or change target but I do not add the level of detail that we have with different reasons for changing…

X: Yes, that sounds good…

[…]

X: What kind of attacks do we describe here?

Interviewer: We mainly look at hacking since that kind of attacks exploits software vulnerabilities. Malware is interesting too.

Hacking, Malware (Adversary Behaviour & Cyber Attacks)

X: So, if we think about hacking, not many software solutions are actually vulnerable because they are simply not public facing assets. Those apps that are connected to the outside world, there the vulnerabilities are really high risk ones… the others are dangerous too but since they are not public, they are less likely to be found and exploited. So, yes, the structure here is completely correct, but I'm not sure how likely that [he talked about attacks] is…

Vulnerability, public facing asset (Defects & Vulnerabilities)

Interviewer: I see. May I ask one question there?

X: *Laughing…* You may… *Laughing…* but perhaps I don’t answer… *Laughing…*

Interviewer: You said that such targeted hacking attacks are quite rare. However, hackers may also abuse a software vulnerability by malware attacks or first enter the organisation by social engineering and then hack internal software that is much less secure. Is that a possible and plausible scenario?

APT (Adversary Behaviour & Cyber Attacks)

X: Yeah… I see where you’re going… Many hackers would not do that because they attack randomly and aim for the low hanging fruits. But if you think those sophisticated guys like state-funded groups, they would possibly do it within an APT.

Low hanging fruit, APT, State funded actor (Adversary Behaviour & Cyber Attacks)

Interviewer: So, I keep everything as it is or do we change something?

X: No, keep everything. I only asked out of curiosity.

Interviewer: Ok, perfect.

[…]

X: Yeah, that is a good point. We need to keep the white hats with us and happy. And if you do not fix fast or if you do not pay as promised or if other things go wrong then you may piss off such a white hat hacker quite a lot and he could attack you or publicly expose the vulnerability.

System thinking, Responsible Disclosure (Security Measure)

Interviewer: Have such cases happened?

X: Yes, there were cases in different industries… One was with a consultancy company that offered ethical hacking and audit to other companies but had a flaw in their own system. Some hackers contacted that company but they didn't believe it and even treated the ethical hackers in a patronising way… Well, they got their revenge and just published the vulnerability… *Laughing…*

Interviewer: Ok, so I guess we keep this here… *laughing…*

X: Yes, keep it in the model…

[…]

X: The workload of a DevOps team should always stay stable. The problem is when it actually stays stable and you do not develop anymore because of all the other work you have to do. Then you delay the stuff on your backlog.

Software Development, Backlog (Software Development), Workload, Delay (Pressure)

Note: ***Quote 5***

Interviewer: So, if we want to know how good something is, we should not only look at the current workload but also at the development of the Backlog you mean?

Software Development, Backlog, Productivity (Software Development), Workload, Delay (Pressure)

X: Definitively! That is quite important…

[…]

Interviewer: Ok, so now we are done with the model. We have checked what is right and what is wrong but now let's step back from it and think about what is missing in the picture…

X: Hm… *Thinking…* Many vulnerabilities can be fixed in the environment… you can for example adjust a firewall much faster and easier than a software vulnerability. So what you do is, you first change the firewall to make yourself safe and then, later you fix the software vulnerability…

Avoid disruption (Software Development), Vulnerabilities (Defects & Vulnerabilities), Environment/Firewall (Technology & Innovation)

Interviewer: Does that always work?

X: No.

Interviewer: Could it happen that such vulnerabilities are then - as you described above - put on the Backlog and stay there and are never fixed because the firewall was adjusted?

Delay (Software Development)

X: It could be but it shouldn't because then the firewall gets quite complex… So, I would say, normally you keep that clear and clean…

Interviewer: Great. Ok. Thanks. Other thoughts? What are we missing?

X: I think nothing, looks good. I think you did a good job.

Interviewer: Thanks X. Nice to hear that. When you consider these insights, what do you think are the most important aspects to improve the software quality?

X: Get mature people, keep mature people. Train them, keep them aware. Test early, fix fast, do responsible disclosure and keep the white hats happy. So if we focus on maturity, we make it right from the beginning. Together with other upstream activities, mature people are most important to security.

Test and fix (Defects & Vulnerability), Maturity, Training (Maturity & Training), upstream activities, responsible disclosure, Keep mature people (Security Measure)

Interviewer: Ok, I see. What about measures regarding attacks?

X: As I said, the firewall is important. External facing assets are really under attack, so you need to have them safe from early on or make them safe when you have a problem. … but actually make it right from the beginning… *Laughing…* Being busy later because of a vulnerability that you could have avoided is quite inefficient. So do it right the first round.

Productivity (Software Development), Vulnerability, Public Facing Assets (Defects & Vulnerabilities), Environment/Firewall (Security Measures)

Interviewer: How do we deal with an attacker? What do we need to know about him or her?

X: An attacker has a business case, like we have one too. [**Quote 30**] An adversary attacks where there are easy gains, low hanging fruits, or great rewards… if you make it harder for him to get in, he will try it at another company, so we are safer…

Low hanging fruits (Defects & Vulnerabilities), Business Case, Adversary Advantage (Adversary Behaviour & Cyber Attacks)

Interviewer: The weakest link among companies…

X: Yes, you could say so.

Interviewer: Ok, do we need to include those things here in the model?

X: No, keep it in mind when you talk about the attacker. But I think the model is good, it matches.

Interviewer: Any final comment X?

X: *Laughing…* no, I think that’s it.

Interviewer: Ok, that’s it. Thank you for your time and effort!

II. D. 7 Unstructured Interview, 6 April 2017

[…]

Interviewer: Who is involved in responding to incidents?

Incident (Adversary Behaviour & Cyber Attacks)

X: It's always those people who are responsible for the software and those people who are needed for responding. At the beginning, many people are involved for finding the root cause. Afterwards, once it is found, only those stay who need to stay.

Response, Root Cause (Security Measure)

Interview: And DevOps are involved here as well?

X: Yes, particularly the seniors among the DevOps. You cannot take a rookie for such a task. You need the best people to solve such an issue.

Productivity, Disruption (Software Development)

[…]

Interviewer: You know that I have worked with the participants on how to measure quality. What do you think? How to measure quality?

X: You look at maturity, training, test results…

Security Measures: How to know about quality -> upstream to downstream ideas (Security Measures)

Interviewer: … you mean the number of defects per feature…

X: Yes, that's correct. Then you look at pentests, responsible disclosure, and in the end at attacks, unsuccessful attacks, and successful attacks.

Interviewer: I see. Why are you so specific with the attacks?

X: The fact that you know about a certain number of attacks, does not mean that there are not more attacks. In addition, you do not measure your quality solely by looking at the number of attacks because you do not have any influence on that number. If for example a malware campaign starts you can be as good as you want, the attacks still hit you. Potentially they are not successful but they hit you. So you cannot only measure your quality by the number of attacks.

Attack, Malware (Adversary Behaviour & Cyber Attacks)

Interviewer: Ok, so we should always look at the three together: Started attacks, successful attacks, and unsuccessful attacks.

X: Yes, correct.

Interviewer: Ok. Overall, how do you value these different criteria?

X: Quite simple: Those that are upstream are more important than those that are downstream.

Upstream - Downstream (Security Measures)

Interviewer: Ok, indeed, quite simple…

[…]

Interviewer: How do you see the connection between productivity and the backlog. People have often talked about disrupting the sprint backlog or putting tasks on the overall backlog. What is you feeling about it.

Productivity, Disruption, Backlog, Sprint Backlog (Software Development), Delay (Pressure)

X: The productivity of the teams is correlated with the backlog

If the backlog grows, we are at least not productive enough to keep it stable. If the backlog grows you may have a strategic delay…

Productivity, Backlog (Software Development), Strategic Delay (Pressure)

Interviewer: What do you mean with a strategic delay? And is that related to agile software development approaches?

Agile (Software Development)

X: Agile is fine with delaying work and that is a good thing to prevent pressure. But if you do that for too long, you create a strategic delay which may cost you, depending on the industry, several percent of your revenues.

Agile, Backlog, (Software Development), Pressure, Delay, Costs (Pressure)

Note: ***Quote 6***

Interviewer: Ok, so that seems to be a potential downside of agile.

Agile (Software Development)

X: Yes, you could say so. And I don't think that this is acknowledged enough throughout all those industries that use agile nowadays…

Misperception on agile (Perception)

Interviewer: Very interesting insights, thank you.

[…]

II. D. 8 Conversation, 6 April 2017

Interviewer: X, Y, do you have time for two short questions? It's about security awareness trainings and the two of you are amongst the trainers for security within the organisation and you conduct these trainings for several years by now, so I thought that you are the right ones to ask.

Security Awareness Trainings (Maturity & Training)

Y: That's true. Yes, go ahead. We have a meeting with Z in a bit, but until then it’s fine.

Interviewer: Great thanks, it’s going to be short. So, first question: Have you ever experienced some kind of “security-fatigue” among the employees within the organisation?

Overregulation (Rival Theory)

X: In general, we do not experience such security fatigue among the trained staff because the organisation is just too large. We do not come so often to people that they feel annoyed by the trainers and the security topic.

Y: Today I had a session with people I have trained the last time half a year ago. The training was still about the same topic, but we adjusted the content in such a way that there were enough new insights for the trainees. After the session, the employees expressed their happiness about the training and were asking when I would come back the next time.

X: Indeed, when W and me announce a new training with exactly the same name, then people will think ‘oh, the two again, I already know all of that’. So, for us it is necessary to communicate that the “same presentation” is actually very different from the one that people have seen previously. We constantly update our training based on new knowledge, and so people always receive new insights from training. So, I have not seen a security fatigue but rather a “meeting fatigue” if people think that the meeting will not lead to new insights.

Overregulation (Rival Theories)

Interviewer: Thank you for these descriptions. Second question: I would like to know how long your trainings generally take?

Y: Training A and similar ones last around one and a half hours.

x

X: Yes, that is correct. Other trainings such as B take three hours.

Interviewer: So, to sum up, all trainings last between one and a half to three hours?

Y: Yes, that is correct.

Interviewer: Thank you very much for your time and your helpful responses.

II. D. 9 Observation with DevOps Team, 26 April 2017

Setting

The researcher was picked up by one of the DevOps. Upon arrival, the researcher was introduced to the team.

The team started on time with the Stand Up Meeting. As the name indicates, the entire DevOps Team (six people) and the Product Owner from the business side were standing around a screen and conducted the meeting. One of the team members was in front of his computer and continued working.

Productivity (Softwar Development) or Time constraints (Pressure)?

Workload / Observational (Pressure)

Round-wise, each DevOp presented his/her current work. The others sometimes asked questions or just listened to the one presenting. One of the team members was leading and guiding the meeting.

The team uses the common software for organising a sprint. Next to “to do”, there are the categories “reject / reopened”, “in progress”, “to verify”, and “closed / resolved”. Everything was well organised and clear.

Content and Situational Observations

Particularly dependencies within the team and more with other teams were pointed out.

Dependencies (Rival Theories)

One of the developers had found errors in a software while merging two solutions. The developer described that he had fixed the errors and had added a test which had not been included until that point. The team discussed whether to use the test (enable it) or keep it disabled. After a very short time, the team decided to look into it later. Several of the team members offered to look into it together.

Observational -> felt like stressed here (Pressure)

The team was subject to a heavy dependency. Several of the items were marked / flagged because of the need of information / work from other teams. For instance, the responsible from the other team planned a meeting on a very short note. The DevOps team seemed to be quite annoyed by such a spontaneous meeting and discussed whether to meet. They came to the conclusion that they would try but probably drop it because of too little time available.

Dependencies (Rival Theories)

Observational (Pressure)

One of the developers described an item that, if not fixed, could not be released. In this case, dependencies played a significant role. The team seemed stressed about that.

Dependencies (Rival Theories)

Additionally, another developer described that a high-priority item (for the overall organisation) was subject to difficulties. He described it as likely that it may not be possible to deploy the software. The atmosphere became more tense, the people were talking faster and more hectic, the DevOps team appeared to be stressed.

Defect (Defects & Vulnerabilities), Stress (Pressure)

The team discussed the availability of each team member and clarified whether enough people would be present on the weekend for the deployment.

Pressure (Pressure)?

Overall Impression

The meeting was governed by stress. The DevOps constantly looked at their watches, postponed several problems to be talked about after the meeting again, and discussed several topics that were very problematic.

Pressure, Stress, Observational (Pressure)

The team pointed to several of their items that, if not fixed and solved, could not be deployed.

Defect (Defects & Vulnerabilities)

II. D. 10 Observation with DevOps Team, 8 May 2017

Setting

The DevOps team was in the same building as the researcher. Hence, the researcher came to the office without being picked up. Upon arrival, the researcher was introduced to the team.

The researcher was a bit too early and received a place to work. During the time prior to the stand up everybody was busy working. At the same time though, the atmosphere was good, sometimes there were jokes and everybody was relaxed.

Productivity, agile (Software Development), self-organising (Pressure)

The two Product Owners from the business side were not present. Instead, they had sent an e-mail which indicated that the team could decide what to do. The team discussed the matter as follows:

“What about our friends from business?”

“They are not here today. They sent a brief email, basically saying ‘do what you want’.”

“What do you mean?”

“”We just re-evaluate the backlog and set something for the next sprint.”

Agile (Software Development), Self-organising (Pressure)

The team started on time with the Stand Up Meeting. As the name indicates, the entire DevOps Team (eight people) was standing around a screen and conducted the meeting.

Productivity (Software Development) or Time constraints (Pressure)?

Round-wise, each DevOp presented his/her current work. The others asked questions or just listened to the one presenting. The team had a good overview of each other's work. One of the team members was leading and guiding the meeting. Another team member - she appeared to be the one with the best organisation and overview - interfered sometimes if necessary.

The way of work appeared to be rather informal and unstructured.

Agile (Software Development)

The team uses another, yet common software for organising a sprint than the previous DevOps team.

Content and Situational Observations

Overall, the DevOps described what they had done and what they were going to do next. The descriptions were at part quite technical and are not listed here for two reasons, first because those aspects are out of scope, and second, because of confidentiality reasons. The following quotes describe how the team talked and communicated.

"I learned something there, was quite cool…"

"Today I pick up a new story and just see what happens…"

Obviously not stressed (Pressure)

One of the DevOps described a vulnerability […] The DevOps had sent a report to the responsible. The conversation went on as follows:

"If they can't suppress it, we don't go to production."

"Why not?"

"Because we do not deploy in such a case."

Software Security (Software Development), Security Risk (Pressure), Vulnerability, Fix (Defect & Vulnerability)

The team talked about the planning of the day regarding the retrospective and the panning. One of the team members could not be present for the retrospective which did not seem to be an issue.

Overall Impression

The overall atmosphere was very relaxed and somewhat even excited regarding the next sprint. The team was in a good mood, no time pressure appeared to be present, and it seemed like the team was satisfied and had a good working relationship. Additionally, the team appeared to be fully self-organised as they freely decided on what to do when and in which order.

Agile (Software Development), Time Pressure, self-organised (Pressure)

II. D. 11 Observation with DevOps Team, 8 May 2017

Setting

The retrospective session started shortly after the Stand Up. The time in between was used to work, whereas the atmosphere was quite relaxed and the people continuously talked with each other and joked.

Productivity, Agile (Software Development)

"…, how is the planning today?"
"Easy planning…"
"Yeah, we just do 68…"
"What 68?."

The retrospective session was started somewhat on time. One of the team members was missing due to another meeting. During the first minutes in the meeting room everybody was joking and the team did not really enter the topic. Once they started with the retrospective, they closed the door and became more serious.

Agile (Software Development) -> interesting as some authors (e.g., Schwaber, 2004) said that closed doors are not necessary…

Retrospective describes the activity of reviewing the previous sprint. Each team member got a pen and post its to write down

See next page.

what went well and what could have been better. In addition, the team members wrote down scores to give an impression of their feeling about the last sprint. One of the team members was collecting and organising the post its on the wall. While there was one organising the meeting, there was nobody who was leading the others.

Agile (Software Development)

Content and Situational Observations

The team clearly emphasised to aim at good quality. One of the DevOps referred to “a period of hot fixes” from which they were out by now but to which they also definitively do not want to go back.

Secure Software Development (Software Development), Firefighting, Business Risk, Security Risk, Quality (Pressure)

The team discussed several more specific content issues which are not listed here for reasons of relevance and confidentiality.

The team discussed the problem of dependencies on other teams, softwares, or even places where they need to travel to.

Dependencies (Rival Theory)

One of the DevOps was really enthusiastic about developing some features. While he did some really important work in the previous sprint which was very much appreciated by the entire team, he was not too happy with it: “I really didn’t know what to do. I was fixing the entire time. I have the feeling that this round was somewhat skewed. Adding features was really not a goal anymore… I believe there is enough on the Backlog to do… I think I would like to develop something in the upcoming time.”

Discussing the quality of software and the review process, one of the DevOps emphasised a point that most others did not seem to be aware about: “I’m new here, so I was really wondering… Why do we not grab the coder and do the code review together. I mean, it’s a great learning opportunity, we should really consider doing that, but I guess we don’t have the time, right?” [**Quote 24**] … Thoughtfully, the others agreed with him on that suggestion. Interestingly, there was not much discussion. One of the others mumbled something about that “if you have way to little time, you don’t have time for such activities” but afterwards the discussion went somewhere else.

Time Pressure (Pressure), errors and vulnerabilities (Defect & Vulnerabilities), Learning (Maturity & Training), Differences in perceptions among employees (Perception)

Quite a statement for time pressure (Pressure)

Within the team, fixing seems to be really appreciated!

Close to the end of the retrospective, the team came back to the topic on dependencies and collaboration. Next to general dependencies (longer conversation about it), some of the DevOps complained about a team that did not fulfil its duties regarding standby. DevOps teams normally have a standby agreement with other teams to cover each other in case of emergencies, in both business and security. The DevOps pointed out the following:

“They [another team] don’t do standby, so if there is an issue we have to work quite hard. I think, we should escalate that to a higher level.”

Note: ***Quote 35***.

Lack of Compliance (Security Measure), Dependencies (Rival Theories)

Hence, in case of an emergency, problems may arise from time delays in response, pressure, or understaffing. The team discussed all options and decided to go the official way to escalate the problem within the hierarchy of the organisation.

Pressure (Pressure), Emergency, Response (Security Measures)

Overall Impression

The DevOps seemed to be quite satisfied with the previous sprint. One of them summed it up by saying “I feel it was productive. Could have been better, but I think it was good.”

II. D. 12 Observation with DevOps Team, 8 May 2017

Setting

The planning of the upcoming sprint was initially planned for after lunch. Due to a short term appointment within the department of the DevOps the planning session was postponed to the later afternoon. The DevOps took care that this would fit the agenda of the researcher.

Either part of company culture or part of agile but all over people where always taking care that things match for the researcher…

In contrast to the retrospective, the planning did not take place in a room but just in the open work space. The entire team (including the one who was absent for the retrospective) gathered around a large screen and started to discuss the work for the upcoming sprint.

Agile (Software Development)

There was no manager or leader but instead the entire team decided together, underlining the self-organised nature of agile approaches.

Agile (Software Development), Self-organising (Pressure)

The planning functions by assigning points to different tasks on the Backlog.

Content and Situational Observations

The planning was quite informal and often based on guesses. The team even joked about it. Yet, there seemed to be some kind of informal roles within the team, meaning that some of the DevOps were pushing for more, others for less, others for different work. It seemed to be quite balanced and productive.

The team made several steps: First they analysed how much work was left from the previous sprint. This work would need to be finished first, and so they had to take those tasks with them to the next cycle. In total those were X points. Next, the

Pull Lean (Pressure)

team discussed the number of tasks on the backlog. They also discussed the importance of those tasks. Finally, the team always referred to their normal number of points they make per sprint to guide their decision process. In short, the decision process was guided by the normal amount of work done in a sprint, by the amount of work that is left from the previous sprint which needs to be done first, and by the amount of work that exists in total.

Push Lean (Pressure)

Anchor (Pressure)

The self-organising and flexible nature of agile approaches became quite clear when the team decided to try to do much more work as usual because of a very important task. Interestingly, the team discussed very long whether they should "risk" to take the task despite believing that it would become too much. Having been asked by the researcher about what could happen if the team could not manage to make all tasks, the team acknowledged that nothing would happen as the products would simply not go to release as long as they are not finalised.

Secure Software Development (Software Development)

Overall Impression

The team decided very freely about the upcoming work. The system is flexible. The system very much resembles the lean approach, as also indicated by participants from the group model building workshops. Yet, some parts of the planning seemed to be too unstructured (e.g., one DevOps mixed the entire time the points that needed to be set for the different tasks). It was obviously very good that one of the team members had a very good overview. Overall, the team seemed not stress but rather excited about the upcoming sprint and the tasks they had chosen.

Lean (Pressure)

Not stressed (Pressure)

II. D. 13 Conversation, 12 May 2017

The following quotation was written down after a private meeting with an Executive from another organisation. The meeting was not related to the study or to the European financial organisation. Yet, the topic of the conversation was cyber security and the comment summed up many previous conversations the researcher had with colleagues.

X: Business is about risk. If there is no risk, there is no business because everybody would simply do it. Cyber attacks are just another kind of risk we have to deal with.

Business Risk, Security Risk (Pressure)

Note: ***Quote 39***.

II. D. 14 Conversation, 18 May 2017

This meeting took place with several colleagues from the European financial organisation and an external expert in software security. The researcher joined the meeting for two reasons: First, the meeting was an interesting learning opportunity, second, he could discuss software vulnerabilities, testing, and ways for improvement. In the light of this setting, the notes below do not represent all aspects discussed during the meeting but only those which are interesting for this study and which are at the same time not confidential.

[…]

X: The global errors in software are piling up. This is potentially not recognised but the overall number is, in my opinion, by far larger than actually reported.

Errors (growing) (Defects & Vulnerabilities); Misperception / Understanding of problem, global, not organisation specific (Perception)

Note: ***Quote 10****.*

[…]

X: I think, developers need better help while making the software. I don’t mean larger handbooks, or longer guidelines because I think that does not help. I want to bring knowledge close to the developers. The developers should know at each very specific stages what to do, and why to do it, and how to do it. With this approach, the developers see how things are going while testing.

Overregulation (Rival Theories),

Direct Support for DevOps (Security Measure)

[…]

X: The global market runs behind. We fix too late. That costs much more and that takes much more time than just doing it right from the beginning. It is also that software is becoming more complex which makes it even harder to fix afterwards. So, what we need is, that we guide developers from the beginning.

Vulnerabilities, Fix (Defects & Vulnerabilities)

Software Complexity (Technology & Innovation)

[…]

X: I think, giving the developers clear help while programming is the best and cheapest and most efficient way to go. And we can think here about automated solutions too. Imagine that the programmers get the information about the quality of the software in real time. That would have a huge impact.

Direct feedback -> System thinking (Security measure), Automation (Technology & Innovation)

[…]

Note: The described approaches would very likely improve the software quality through a simple approach, particularly the automated real-time feedback! Yet, it is unclear to what extent aspects like compliance, pressure, lack of awareness, or unknown problems are considered in the discussion. This remained open at the end.

Pressure (Pressure), Lack of Awareness (Maturity & Training), unknown defects and vulnerabilities (Defects & Vulnerabilities), Compliance (Security Measures)

II. D. 15 Unstructured Interview, 18 May 2017

Interviewer: Hey X, thank you very much for taking the time.

X: No worries man, I really hope that I can help you.

Interviewer: Thanks, that is kind. I am pretty sure you can. Let's start quite easy [*smiling*]; Why do we have vulnerabilities in software?

X: *Laughing…* Yes, that is a very easy start… To be honest, I would say it's business. You know, for business you look at functionality, at features, at availability, and so on. As long as that is given, business is happy.

Business risk (Pressure)

Interviewer: I can imagine. Anyways… What do you think, is there a way to find out how many errors you have on average per feature?
[It followed a very long discussion. In the end, it seems that there is no way of knowing that. What is known though is the number of defects per line of code. If somebody can find out the lines per code per component and than the number of components per feature, then it would become clear how many errors exist on average per features]

See below, II. D. 16

Interviewer: Anyways, I will try it later agin with some others or with the DevOps team. And if we don't know, we don't know, then we can also not make a number up.

X: There you're right. But man, I'm sorry that I couldn't help you out there.

Interviewer: No worries.

X: Can I help you with something else? I still have quite some time to kill before the next meeting. *Laughing…*

Interviewer: That is great, thanks. Yes, I guess you can. I actually wanted to know the numbers to have a better idea about how many vulnerabilities arise from defects in software that is released.

X: That is a good question… I can really tell you, vulnerabilities are not only critical because of their level, but also because of the underlying mathematics. The numbers count because many low and medium vulnerabilities may be as dangerous as one or two critical ones.

Vulnerabilities, Criticality (Defects & Vulnerabilities)

Note: ***Quote 28****.*

Interviewer: Yes, I can totally see that. I was quite surprised when I got to know that in many industries companies simply ignore the lower vulnerabilities and put them on the backlog…

Delay (Pressure), Vulnerabilities (Defects & Vulnerabilities)

X: Yes, that’s it man. You know, it is like meta data. One piece alone is really not helpful but combined, many lower vulnerabilities can give an attacker quite some information. They enforce each other and they function like stepping stones, it’s like a domino effect.

Vulnerabilities (Defects & Vulnerabilities)

Interviewer: That sounds quite logical to me I have to say.

X: Yes, I think so too. But you still have so many low hanging fruits out there… Many companies underestimate their real weakness and many of them have a very poor error handling.

Weakness (Defects & Vulnerabilities), error handling (Security Measure), Low hanging fruits (Adversary Behaviour & Cyber Attacks)

Interviewer: X, we started with that at the very beginning, but let’s come back to that one more time. How is it possible that we have so many software vulnerabilities in this world. Which role does stress play in this matter?

X: Stress is really important. You can see from the code whether somebody was stressed or not. You simply see it in the quality.

Stress (Pressure)

Interviewer: What else may be problematic for software security?

X: You know, when teams plan their sprints - I know that from experience, I also worked as a DevOps - it is easy to plan with functionality. You have a “proof of concept”. Once your software functions, well, then it functions, and you stop testing it. Such use cases cannot be employed for security. The problem is, many programmers are used to stop after one successful test. For security you use, however, abuse cases where the main idea is to find as many problems as possible to make the software secure…

Use / Abuse / Software Security (Software Development)

Interviewer: But this problem seems to be known… why do DevOps not simply spend more time on that? I mean, there is no reason to stop after the first “successful test”.

X: That is correct… but DevOps often lose the fight against the product owner who prefers functionality over security… *Note:* ***Quote 15****.*

Business, Financial Focus (Pressure)

Interviewer: So you mean, in the end, the business side has the last word.

X: Kind of, yeah. What you also have to see with tests: Everybody is afraid to miss something, so that increases the number of false positives. People rather report too much than too little. But when you continuously report really high numbers then your mind is simply not able to deal with all these reports. And after some time, your good efforts to avoid problems create problems because you make errors. Perhaps we can even say, the higher the false positives, the higher also the false negatives [i.e., missed vulnerabilities].

Stress (Pressure), Test (Defects & Vulnerabilities), Systems Thinking (Security Measures)

Note: Very interesting concept. Since positives / negatives are barely considered in this study, here potentially not applicable. Yet, X describes a very interesting mechanism.

Interviewer: Vulnerabilities come from errors…. So, how much effort does it take to actually fix an error?

X: That really depends. Complex errors may take lots of time, typos can be solved within less than ten minutes.The problem is when you have technical debt. There you can say, the older the worse and the more difficult to fix. If you wait for instance several years fixing that software becomes obsolete… In my opinion technical debt and the lifespan of a software are highly correlated. You know, it is so stupid… if you need to fix a bug from the previous sprint, it takes you in maximum an hour, most likely rather a few minutes. If a bug is a year old, you may need weeks. To avoid that I think, refactoring should be done on a regular basis as part of the normal development process. Sadly, in so many companies it is never done.

Software Complexity (Technology & Innovation)

Error - Technical Debt (Defect & Vulnerability)

Agile (Software Development), Lack of compliance (Security Measure)

Note: ***Quote 21****.*

Interviewer: What do you think, why is it never done?

X: People often say ‘we do it later because now we really do not have the time’, but then later it is simply not put on the Sprint Backlog and just not done. [**Quote 27**] In addition, if you plan with refactoring you already have a problem… as I said, you should simply do it on a regular basis… Don’t misunderstand me, I don’t want blame the DevOps, often you also have dependency problems in agile and then you can suddenly never do refactoring because you have to wait for others…

Time pressure, delay (Pressure)

Dependency (Rival Theory)

Interviewer: Yes, I have already heard quite often that dependency is a hue problem in agile… But X, I think you’re meeting is in a bit. What would be your final comment for this very nice and enriching discussion?

X: Oh, yeah, you’re right man. Hm… I really like the kiss approach. You know, keep it simple and so on… and I think if stress is high, quality and particularly security go down which costs a lot in the long run. In fact, cheap is always expensive in the long term. Eventually, you need security anyways, so do it right from the beginning! When you do it later, it costs more, it is harder, and it harms your business.

Stress, Costs, Risks (Pressure)

Note: ***Quote 38***

Interviewer: Hey X, thank you very much that was great, you gave me so many valuable insights.

X: No worries man, I really enjoyed it. Let’s continue another time. And if you have some questions in the meantime, just come over…

II. D. 16 Conversation, 18 May 2017

This conversation took place with a security expert who was on a short visit in the office as he normally works in another country. The expert and the researcher discussed many technical details of software development and software security. The discussion was active and resembled rather a workshop than a mere conversation. The text below summarises the gained insights which are suitable for documentation and gives an impression of the work. Next to the discussion given below, the expert and the researcher also checked data [particularly numerical| and analysed it together to make sure that the right insights were used for the study and the project within the organisation [for confidentiality reasons not shown below].

[…]

X: Y, lets get some food an then get to work!

Interviewer [Y]: Great, let's go. [around 17.00h]

[…]

X: So what do you mean with an error? What is an error for you?

Error, Mistake, Defect (Defects & Vulnerabilities)

Interviewer: Any kind of mistake you have during coding of configuring the software. Or more generally. An error or a defect is anything you do not want to have in your software.

X: Ok, quite broad definition but I see the value of it. For security, all those things may be a problem.

Secure Software (Software Development)

Interviewer: Exactly, that is why I chose such a broad definition.

X: Good for me! Now let's see. You know that most software vendors develop their software with libraries… let's go into that…

Technical Solution (Technology & Innovation)

[Shortly later…, around 17.30h]

X: So, if you have an error, or defect, or mistake, or whatsoever you want to call it in this library, is that one mistake for you or are those five mistakes because five features are based on that library.

Technical Solution (Technology & Innovation)

Interviewer: Tricky question… Well, you need to fix one mistake but - assuming that this mistake makes you vulnerable - you have five vulnerabilities. What do you think?

Error, Mistake, Defect, Vulnerability (Defects & Vulnerabilities)

X: Yes, I think that sounds logical… Ok, let's get to the laptops and do some research on open source software… we should be able to find there the number of defects per feature…

Interviewer: Great, let's go.

[Quite some time later…around 18.15h]

X: I think it should be possible to find out how many errors you have per feature. We can find online open source solutions or the software of vendors and check there. We already know the number of errors of lines per code [1-10/1000 is seen as worldwide average]. We now need to find out the actual lines of codes for a software and then we need to see how many classes and how many components such a software has.

[Quite some time later… around 19.00h]

X: It took more time than expected, but here we have something that brings us closer… based on this open source solution we can expect to have […] files for a class and […] files per component.

Interviewer: How do we get closer with these insights to the number of errors?

X: Let's see how we can make that…

[Quite some time later… around 19.30h]

X: It really seems that there is no such direct relationship… Do you have any other idea or hint how we can get there from your previous investigation?

Interviewer: Well, initially I thought you would answer me that question! *Both laughing…* I have one paper [Rahmandad, & Repenning, 2016] that gives a number but I am not sure whether it is accurate… and I will meet a DevOps team and this is definitively one question that I am going to ask…

X: So, what was the number?

Interviewer: Tell you first what you would guess and then we compare. *Laughing…*

X: Puh, that is hard… For this open source software we looked into… If I have to guess I would say it is definitively higher than 0,3 defects per feature… what does the paper say?

Secure Software (Software Development), Error, Mistake, Defect, Vulnerability (Defects & Vulnerabilities)

Interviewer: 0,5 per feature… so could even be an accurate number… Well done! *Both laughing…* I will check with the DevOps…

II. D. 17 Unstructured Interview - Validation 2, 23 May 2017

These notes document an unstructured interview that evolved from the presentation and discussion of the outcomes of the study / project to the involved team in the security department of the European financial organisation. The presentation was initiated by the responsible for software security who wanted the team to know about the project in greater detail. While the presentation had an informatory function for many of the experts, for the purpose of the study this presentation served as second opportunity for validating the findings of this research. As described above in II.D.6, the researcher used the approach of disconfirmatory interviews for validating the model and the results. In contrast to the recommendations from Andersen and colleagues (2013), but in line with the wish of Diker (2003), the disconfirmatory interview was done with a group and not in an individual setting. For further information see below at II.E. Prior to presenting the model, the researcher strongly invited the participants to question the results and interrupt the presentation whenever they felt like. Once more, the model was shown to the experts by unfolding it feedback loop per feedback loop. Due to the nature of this meeting, the text below does not show the entire interview. Instead, it provides the notes taken by the researcher during the presentation and discussion. Without being experts, the team members were all familiar with system dynamics terminology and methodology as they had been confronted with several times.

[…]

X: But does this model include automated testing?

Automated Testing (Technology & Innovation)

Interviewer [Y]: Not explicitly…

X: But everybody uses automation nowadays… what does a model serve us if we do not have automation in there?!

Z: Well, I would say we have automation in there. Simply because it does not need an own structure. It is the same thing for the DevOps, just faster and better…

W: I agree with that… Automation just means that we do our tests with automated tools, but it does not say that everything changes…

X: Wait, wait… I cannot follow… why do you talk about testing? And why does nothing change with automation?

V: X, I have to say that W, Y, and Z, are right. Automation is simply for testing. Yes, there are a few other things we can do with it, but for now this is limited. It safes us lots of time with testing and gives better findings than many manual tests, but we still need to fix it ourselves. Isn't it W?

Test, Fix (Defect & Vulnerabilities), Automated Testing (Technology & Innovation)

Note: ***Quote 11****.*

W: Yes, that really is the case! Discovering but not fixing doesn't help.

Secure Software Development (Software Development), Test, Fix (Defect & Vulnerabilities),

X: How is that possible, and why is it done so much then?

W: Because it really helps. You find much more defects with much less work. That is great.

Z: So X, I would say, we cover automation here quite well. I think we can keep it like this in the model.

X: Hm… Ok, let's move on.

[…]

U: Do you mean with awareness that the DevOps are aware of the training possibilities or that they are aware of security or doe you mean both?

Awareness (Maturity / Training)

Interviewer: Good question! Yes, we had this discussion previously. The model here only shows the awareness of DevOps for training and then whether they are trained and then whether they are mature. Since security awareness is part of maturity, we thought about including it here. What do you think?

Security Awareness, Maturity (Maturity & Training)

U: Ah, yes, I see. Yeah, it's not so intuitive at first glance, but I get what you mean.

Interviewer: Ok, thanks. Do you think we should change it?

U: Ah, no, I think it's ok like that…

T: I think so too. But we should make clear when we work with the model in the future and present it that maturity includes awareness because creating an awareness culture is really one of the major solutions for security.

Awareness Culture (Maturity & Training)

U: Yes, I agree. We should keep that in mind.

Interviewer: Thank you for the comment. I will take a note on that! Still not change it?

U: No, I think it's fine.

[…]

S: Do you cover with that attack also insider threats?

Insider Threat (Adversary Behaviour & Cyber Attacks)

Interviewer: No, not really. I mean, if an insider attacks like an external, then yes, but else no.

Z: I would also say that this is outside the boundary of this model. An insider really works differently than what we see here. An insider has more motivational aspect about why to attack.

U: Yes, I agree with you but I could imagine that to a certain extent this structure holds for basically any attack. Any attacker needs to have information, find a way in, and actually exploit that way. So, yeah, you say its different, but the way of the attack is quite similar for me…

Attack (Adversary Behaviour & Cyber Attacks)

Z: Yes, that may be. But I think an insider attack has further dynamics that we do not look at here.

Interviewer: S, is your question answered?

S: Yes, perfect, I just wanted to know that…

Interviewer: Do we need to change something or adjust something or do we need to keep something in mind here?

U: No, I think all of us got that this is not an insider threat but I think also that this attack structure is quite generic, so its good. Keep it.

Z: Yes, keep it like this.

[…]

S: Why do you have the footprint here? What do you mean with that?

Footprint (Security Measure)

Interviewer: Well, the larger an organisation the more known it is and the more likely it is a target. So an organisation may get attacked simply because it is known or because it is part of a certain industry, and so on…

V: I agree… I think the footprint is quite important. There are even organisations that do social media data mining to have intel that they can use for preventing attacks.

Social Media Data Mining, Prevention (Security Measures

Interviewer: Clear, question answered or shall we adjust something?

S: Nono, fine…

[…]

R: Are those DevOps internal or external?

Interviewer: For now, we focused on internal DevOps. But yes, I am aware of the trend that many companies outsource their people and hire instead external providers.

DevOps (Maturity & Awareness)

R: Yes, that is correct. It's quite common.

Q: But I think it's interesting. External people are often more mature because they are experts in their field and you hire them

See next page

for that. But I'm not sure whether you lose that benefit because those externals do not know the culture and system of the company they are working in…

Maturity, Culture, System (Maturity & Training)

T: Yes, I agree on that.

Interviewer: Ok, do we need to make that explicit here?

Q: Na… I don't think so. R?

R: No, it's fine like that.

[…]

X: But that is unlikely… How many people does an organisation need for handling incidents?

Interviewer: I completely see that this is unlikely. But it exists and thus offers the potential to firefighting attacks.

Firefighting (Pressure)

Q: I think this is a really interesting and valid mechanism! And I think we should be aware of that. Just recently, [another firm] had something similar. They had more than a thousand people in the warroom for an entire day. This was a planned exercise for training but the business impact was still heavy. I can really imagine that this has a strong effect on companies.

Firefighting (Pressure)

Note: ***Quote 34***.

[…]

Interviewer: Are there any final ideas, comments, remarks, wishes? What do you think is still missing here?

S: Do we really have overregulation?

Overregulation (Rival Theory)

Interviewer: Sorry, I missed to point that out. This structure shall only indicate that you can create overregulation, not that you have it…

Q: *Laughing…* With regard to one specific thing [name deleted] we definitively have overregulation *Many laughing…* But no, I think its perfectly fine.

S: Ok, yes, good.

Interviewer: Are there any other questions or comments?

X: How do you make sure that you have the right data? Yes, this one is qualitative but when you build the model, I mean, the simulations, how do you know that you have the right data.

Validity (Future Research)

Interviewer: I rely on the help of this team.

Z: Yes, we work together on getting the right data. We check that we have the correct numbers when we get information from others.

Validity (Future Research)

X: Make sure that you really do that. Otherwise we cannot use the insights from the simulations later on…

Validity (Future Research)

Interviewer: We use very clear validation methods…

Z: Yes, we check our results with historical data, with expert opinion, with publicly available and internal data and so on…

Validity (Future Research)

W: X, we really make sure that the data is correct. Y and I spent last week more than two hours on discussing errors and mistakes and vulnerabilities and checked that we have the correct data. We're really on track there.

Validity (Future Research)

X: Ok, that sounds good. Guys, I just want that we do good work. But it looks like we do, I like that.

[…]

II. D. 18 Conversation, 30 May 2017

X: Good morning, shall we grab a coffee?

Interviewer [this was really a conversation, so the term does not really apply but it is kept for consistency]: Yes, of course, let's go.

X: How is the project, the model, and the study going?

Interviewer: Thank's for asking quite good. I really liked the recent team meeting. That got me quite some ideas.

X: That is great. What did you like about it? What are your thoughts about it?

Interviewer: Well, I found the discussion on automation really good. It was quite interesting to see how V and W pointed out the limitations of it. For me it really seemed like that you lose all benefits of automation if you do not fix after finding. That was really interesting and I also had the impression that there may be some misperceptions about the actual strengths of automation… What do you think?

Automation (Technology & innovation)

Fix (Defect & Vulnerability)

Understanding of Situation (Perception)

X: Yes, true. People want automation but finance basically says 'you get automation, but how many people do we save by that?' The problem there is, even with automation we do not get better. It is true that we detect more but since we have less people, we can't benefit from our increased knowledge because we can't fix it. *Note:* ***Quote 12***.

Business, Finance, Risk (Pressure), Layoff (Maturity & Training), Automation (Technology & innovation)

Interviewer: Yes, I really have the impression that the value of normal staff is often underrated. I start to have the perception that our initial idea for the modelling of high workloads, higher work efforts in the light of too little people, and errors seems to be quite valid.

Workload, Pressure, Understaffing (Pressure), Layoff (Maturity & Training), Error (Defect & Vulnerability)

X: Yes, I think so too.

Interviewer: Not saying that it is the case here in this organisation, but it seems like a general possibility.

X: Yes, we're doing quite alright. But still, instead of hiring more DevOps for actually doing more work, they hired somebody basically holding a whip for making the DevOps work faster and to increase the pressure to deliver. [**Quote 8**]

Pressure (Pressure), Opinion about management (Perception)

Interviewer: *Laughing…* If you say so… but I see one major benefit from automation even if you do nothing afterwards… at least you know your future work and you do not adapt to a wrong future by underestimating the true number of problems…

Systems Thinking (Security Measure)

X: Yes, you're right. But you could still have to many vulnerabilities.

Interviewer: That's why you need both: People and automation.

X: Exactly, I think we're on a very good track!

II. D. 19 Unstructured Interview with DevOps Team, 2 June 2017

Setting

The researcher visited again the same DevOps team which he had observed during the retrospective and planning meeting. This meeting was at the middle of a sprint, in contrast to the previous ones which indicated the start of a new sprint.

Since the team had offered its help whenever needed, the researcher gladly accepted the invitation to discuss a few aspects most likely DevOps know best about.

The meeting took place in a room. The door was closed. The meeting started exactly on time and ended exactly on time. The atmosphere was from the beginning much more tense than during the previous meetings. Everybody seemed to be somehow slightly stressed.

Agile (Software Development)

Stress? (Pressure)

Content and Situational Observations

Interviewer: Thank you for having me again. I really appreciate your help and you have given me already many insights. Thanks a lot! Today I have quite some questions, so should we just get started?

X: Yes, go ahead.

Interviewer: Can it happen that all of you are working on an incident and nobody is left for developing and operating software?

Y: Well, every team keeps space for incidents. But I think you never have everybody in response.

X: I'm not sure I agree with you… Think about the one case.

There we were quite busy. [To the interviewer] That was not a security incident. That was a planned thing… we decided together with the managers that we want to do something important… we are glad that we did it but there we had months of heavy work… That wasn't really response but all of us were busy…

Workload (Pressure), Incident (Adversary Behaviour & Cyber Attacks)

Y: Ah yeah… you may be right…

Interviewer: Ok… Hypothetically speaking, what do you think are reasons for a growing backlog?

Dependency (Rival Theory)

Z: Dependencies…

W: The items are just not important anymore and nobody deletes them from the backlog…

V: I feel that we do not have too much work, so I don't know…

U: Yes, that's why he said "hypothetically"… *laughing…* I also think that it is dependencies…

Dependency (Rival Theory)

T: Sometimes it is also that things just pop up… you did not plan with it but then something happens and you have to postpone things… I mean, that's also part of a backlog…

Software Development (Software Development)

S: It's also that you have more features which means that you have more work and more knowledge about more work…

Software Development (Software Development)

Z: Yeah… you see… it's difficult… you know, that's the nice thing with agile - nobody knows what you do, not even we know it… *Laughing…*

Agile Software Development (Software Development)

Interviewer: *Laughing…* Ok, thanks, those were quite some answers… Next, do you have fixed tasks or can you change freely between, let's say developing, testing, and fixing?

Y: We move freely.

Agile Software Development (Software Development)

Z: Yes, that's part of agile.

Interviewer: Ok, great. Next question… To what extent do you use automation and how much does that change your work?

Automation (Technology & innovation)

Y: Of course, we use automation. Everybody does that…

W: One of us used to spend his entire time on testing the developed software. Now with automation, we have much more time for developing and operating. Of course, we still do manual testing but by far not as much as we used to. [**Quote 9**].

Productivity (Software Development), Automation (Technology & innovation)

X: Just as a note, we have not too much time left and you said you have more questions. We should hurry up.

Interviewer: Ok, yes, thank you for reminding me. Just based on your experience, how many errors per feature do you think exist?

[Not listed due to confidentiality because here the discussion was not only about open source software that everybody has access to but about internal software, software from vendors, and open source software. Thus, this part is not given.]

Disruption (Software Development), error (Defect & Vulnerability)

Interviewer: Thanks. When do you fix errors?

T: Generally immediately.

Interviewer: Ok, and do you do refactoring to avoid more complex problems?

Agile - Refactor (Software Development)

Y: Ah, we do it sometimes…

U: I think we should take the time for doing it… we do it too little…

Understanding of situation (Perception)

X: Why should we take the time for refactoring? It is obviously done enough because otherwise it would be a priority on our Sprint Backlog. [**Quote 21**]

Understanding of situation (Perception)

Interviewer: Hm… ok… What do you think are main sources for vulnerabilities? And how do you make sure that there is no vulnerability when you release a feature?

Y: There you should ask T…

T: *Smiling…* more and new technology like libraries can really help you but you can also make quite some mistakes. We are really aware of that and take care to avoid problems there… yeah… how do we know about vulnerabilities… generally pentest tells us about the quality of our work.

Vulnerabilities (Defect & Vulnerabilities), Awarness (Maturity & Training), Systems Thinking (Security Measure), Technical Solutions, Libraries (Technology & Innovation)

Y: Yes, I agree. When you pass pentest, you're fine.

Pentest (Security Measure)

X: Yes, I think so too. There are no vulnerabilities in our software because we would know whether the features are vulnerable because pentest would tell us. Since that has not been the case, there are no vulnerabilities. [**Quote 22**]

Understanding of situation (Perception)

Interviewer: Ok… Two last questions…

X: Sorry to interrupt, but I stop here at two o'clock, then I need the room for another meeting…

Interviewer: Ok, sorry, I'll try my best. First, I already talked with S about it, but what is the connection between the overall number of features and the work you have to do.

Y: Well, if there are more features, there is more work.

Software Development (Software Development)

S: That's what I said too. But I also think that when there is more work, we create more features…

Y: Hm… yes, I think so… sounds good.

Interviewer: Second…

X: Sorry guys, it's two and we need to stop. Thanks for coming over and working with us.

Time Pressure (Pressure)

Y: Ah, that's bad. If you need anything, just get in contact with S and come again.

Z: Yes, I think that's good. Would be great.

Interviewer: No worries, thank you for the great help.

Overall Impression:

The team was much more stressed during the sprint. Outside of the room, the researcher continued talking with S who admitted with a grim smile that those days the team was quite under busy.

Observational, stress (Pressure)

II. D. 20 Unstructured Interview, 21 June 2017

Interviewer: X, do you have a moment for me so that I can ask you one question?

X: No… *smiling and silence…* but two is fine for me [smiling]. Sorry, I am in this mood today… *laughing…*

Interviewer: Perfectly fine for me *smiling…* and thanks a lot, X. Do you remember our meeting a few month ago about the topic of software vulnerabilities?

X: Yes of course, we talked about vulnerabilities, how they are created, how we can find them, and how they may be exploited. You have the data on that from Y., correct?

Interviewer: Yes, that is correct. That data is, however, concerned with […], not with the vulnerabilities created by software developers here in the company. Instead, I am particularly interested in software vulnerabilities caused by internal development.

X: Ah yes. Hm… I think I told you that also software vulnerabilities can be exploited by malware as well as hacking attacks?

Vulnerability (Defect & Vulnerability), Hacking, Malware (Adversary Behaviour & Cyber Attacks)

Interviewer: Yes, you did.

X: Ok, great, so we are on the same page. But sorry, what is it that you want to ask?

Interviewer: Ah yes, indeed. I am interested in why there are obviously avoidable software vulnerabilities. According to CVE and the Rand report on Zero Days - you know that one, don't you? [X shows that he knows it] - there is a growing number of known and unknown vulnerabilities. Additionally, Verizon shows the growing number of successful attacks, literature, practice and media emphasise the importance of hacking attacks, and there are support mechanisms such as the BSIMM or OWASP Top 10 to avoid defects and subsequent vulnerabilities. Yet, most of the discovered vulnerabilities for instance by pentests or responsible disclosure, are simple, easy to avoid, and are sometimes even on the OWASP Top 10. What do you think, what is the reason for that?

X: Indeed, I agree with you. Most of what is found is very simple and easy to avoid. I think there are several reasons.

Interviewer: Do you have anything specific in mind?

Business (Pressure)

X: Security is not the major focus of software development.

Interviewer: Do you think about the trade-off between functional and non-functional aspects [Note: in software development and even more general cyber security, it is common knowledge that security always involves trade-offs]?

X: For example.

Interviewer: Since we now touched upon this topic. Thinking from a management or business perspective: how would you see the time dimension in this context? I mean, companies work with short-, medium-, and long term perspectives. How would you describe these two - functionality and security - aspects?

X: Functionality is short term, definitively It’s about creating immediate business value. And it’s about prestige… You know?

Short term functionality, finance, value (Pressure)

Interviewer: I think so, but please, go ahead.

X: Functionality brings the team more prestige here within the organisation. Nobody sees security. So, they go for functionality and for the prestige. [**Quote 20**]

Opinion about others (Perception)

Interviewer: What about medium and long term?

X: A high quality is always nice but I would say it pays particularly in the long term because you build on it.

Long term quality / security (Pressure)

Interviewer: Coming back to my initial question, what is the reason for avoidable mistakes?

X: When developing functionality, we know the use cases. Security is more difficult because we must think of abuse cases. When we need to decide, often the certain use cases come first, and then the uncertain abuse cases. [**Quote 16**]

Use, Abuse (Software Development)

Interviewer: Ok, what else?

X: Security awareness. For many people in IT and business it's simply not a topic. And people underestimate the problem because they cannot imagine the causes and consequences. People are not hired for security but for functionality because security does not pay off and is invisible. If you have no attempted and prevented attack, security doesn't pay off. That's why secretly quite some people in the security field always hope for small incidences, so that managers stay aware of the problem.

Security Awareness (Maturity & Training)

Misperception of Danger (Perception)

Understanding of Security (Perception)

Interviewer: I see, very nice, thank you for the explanations.

II. D. 21 Unstructured Interview, 22 June 2017

The interview evolved from a conversation that the researcher and the expert had anyways.

Interviewer: So X, what do you think. Are security investment rather short-term, or medium-term, or long-term?

X: Ah man, you know, security is always good because when you increase security you increase the overall quality of a software… but many people don't get that.

Secure Software Development (Software Development)

[A colleague joins the conversation… short interruption, small talk and then the discussion goes on].

Interviewer: I see. Ok, if I frame it differently: When we think about software functionality and software security, where do you see those two from a temporal perspective?

X: Yeah… Functionality is short term… you do it now for surviving market pressure.

Short term Functionality, market pressure (Pressure)

Y: I'm not fully convinced there. I mean, yes functionality is short term but I'm not sure whether we have such market pressure… I would say, since we are a financial organisation time to market pressure is less of a problem for us. We are not an app developer. We do not lose market share if we release something later. We also do not have to address all customer demands, but instead make sure that the software that customers use actually functions in a proper way. [**Quote 4**]

Skeptical about time to market pressure (Pressure)

X: Yeah, you could say so… that is valid…

Y: For being good you do not need to be a first mover. Just be an early adopter: You are still fast on the market but you learn from the mistakes of the others…

[The conversation continued, after a while Y left again.]

Interviewer: Ok, I see that point. What do the two of you think about security? What time horizon do we have there?

X: I would say security is rather long-term. I mean in the first 24 to 168 hours you may get attacked more because hackers want to “test” your new software… but afterwards it is really long-term. If you are vulnerable the question is not if you get successfully attack but when. So, I always say: You need security anyways, just build it in and make it right from the beginning.

Short short term with security and then long term (Pressure)

Secure Software Development (Software Development)

Interviewer: What would you recommend?

X: Always check for technical debt, always refactor, invest in good tools and libraries, and do not postpone too much because then you may end up never doing it… I would bet it pays off very fast if you make a good job at the beginning.

Software development (software Development), technical solutions (Technology & Innovation)

Interviewer: All of that makes sense… But then I’m still wondering, why do people not build it in? It seems so much better!

X: Ah man, you know… If there is high business pressure, you go for functionality because you need to survive business. You should not do that for too long though but most of the time you would. [**Quote 19**]

Business Pressure (Pressure)

Interviewer: Ok, so again the business-security-nexus… Thanks.

II. D. 22 Conversation, 22 June 2017

X: Functionality is for making money now and security is a long-term investment. People don't think that far ahead…

Functionality short term, security long term (Pressure), Problem Understanding (Perception)

[…]

X: Why should you knowingly accept such security risks? That's the idea of risk… There was this car case in the USA, quite some time ago, where a company built cars with a tank in the back of the car knowing that people would die in case of a rear-end collision. They still did it because they saved a few dollars per car…

Risk approach to finance (Pressure)

Interviewer: And?

X: They got fined in the end after some people suffered from severe injuries and others even died…

[…]

X: Yes, that is nice about agile. You are more flexible and you try to really have no unnecessary pressure…

Agile (Software Development), Pressure (Pressure)

Interviewer: How do you do that next to make people work harder and longer?

X: One approach is having more people for a project… If we want to increase the people working on something, we do not increase the team size, but add new teams. Let's say we have a team of eight people. We then split the team in two teams with four people each and add four new DevOps to each team, and if necessary, we just continue like this. [**Quote 7**]

DevOps (Maturity & Training), Problem Awareness (Security Measure)

II. D. 23 Unstructured Interview & Validation 3, 22 June 2017

Interviewer: Hey X, thank you for always taking the time for me. These conversations really help check whether I am on the right track.

X: Yes, that's alright. I like our conversations.

Interviewer: Me too, thanks. X, you know, I'm wondering how it is possible to have vulnerabilities in the light of all those efforts to avoid them. I mean, there are process models, there are guidelines, there are automated tools, etc. I heard about a lack of compliance, I heard about lacking awareness, I heard that business does not really care about security because it does not give but cost money, and so on… What do you think, what are the main reasons for vulnerabilities?

X: First of all, I would say that the more software you have the more exposure there is. When you have more assets, then the attack surface is larger.

Weaknesses, Exposure (Defect & Vulnerability)

Interviewer: I understand. So are the numbers of vulnerabilities growing because of the increasing use and importance of technology?

Change of future and technology (Technology & innovation)

X: Yes and no. Yes, as I explained above but also no because the picture is more complex. Many organisations outsource the development and operations of minor, less important issues. Those are generally not connected to the actual organisation, so you do not really risk something. Quite often, the standard within companies is really high and the standard for those outsourced solutions is quite low. I don't like that because an organisation's reputation can still be affected by a successful

Business (Pressure)

Employee Opinion, Managerial Opinion (Perception) attack against an outsourced solution. I mean, the thing is really not important so you can do that but I find it's the wrong approach.

See above…

Interviewer: Ok, I see that part. What else do we have? You know, I have now so much data and information that I want to consolidate the insights. But for doing that I need to be sure that I didn't miss something.

X: Clear. Another thing is whether you talk about public facing assets or internal assets. What you could see with the massive, publicly well documented hack against a large electronics, entertainment, and movie company where an attacker managed with hard work to get insight their system. Afterwards, however, he could move around entirely free because their internal assets were really not well protected. You don't want that…

Public facing assets (Defects & Vulnerabilities)

APT (Adversary Behaviour & Cyber Attacks)

Interviewer: Yes, I have heard that before. But of course, if you have limited resources, you first defend on the outside…

X: Obviously, that's why such situation occur. You said earlier that there may be a lack of compliance… I'm not sure I would follow you there. I think it depends on your reference point. Some industries have really high standards, others have lower ones. Thus, somebody can be very compliant and still develop very insecure software… and somebody else can develop very secure software but not comply to rules…

Compliance (Security Measures), Understanding of Situation (Perception)

Interviewer: That is very interesting. Thank you. Yet, I heard quite often people talking about this topic… they did not seem to have such a balanced view as you just described…

Anyways… Somebody once told me that the tension between functionality and security is as normal as the tension between freedom and security in our society… what do you think?

X: I fully agree. It is normal, it should be there and we can handle that.

Business Risk, Security Risk (Pressure)

Interviewer: But if you have this tension, you have stress… haven't you?

Business Risk, Security Risk, Trade-Off, Stress (Pressure)

X: Yes, true. But also the other way round: If you have no stress you have no trade-off.

Interviewer: How do you think people generally decide when there is this tension?

X: If business is important, people take a shortcut with security or simply postpone tasks. It stays on the backlog though, so it should be done… you cannot just take something away from the backlog without doing it, so it should be done…

Business Risk, Security Risk, Delay (Pressure)

Interviewer: Do all teams follow that rule of "business-first"?

X: *Laughing…* No, by far not.. It depends on their maturity and awareness about the topic. The more mature, the less likely are they to drop any security issues… The good thing in agile is that we have an ownership culture. The one who builds and the one who manages is responsible. That means, if something goes wrong it is first the business side that has to explain it and then the DevOps… so in that sense there is quite an incentive to make things right…

Agile, Secure Software Development (Software Development), Maturity Culture, (Maturity & Training)

Interviewer: I understand… let's move on to something else… How would you see functionality and security from a temporal perspective? Are those two short-, medium-, or long-term?

X: Functionality is short term, that is clear. You create functionality for creating value. Security is also somewhat short-term but also long-term… I think, it's rather a long-term thing… The problem here is that long-term benefits from security are sacrificed for short-term gains from functionality because people think it is so unlikely that something happens. [**Quote 17**]

Short term Functionality, Long term security (Pressure)

Trade-Off (Pressure), Opinion and Understanding about Situation (Perception)

Interviewer: And?

X: Ah, they are wrong, obviously!

Interviewer: Ok, I guess I have it. Do you have some more time?

X: Of course, let's go through everything you need.

[The interviewer showed the expert the generalised causal diagram from Figure 15 above. Then the interviewer explained the generalised diagram and asked the expert - as explained above and following the same rules - to disconfirm and criticise the model. Since the expert already knew this process, he fully took on this role.]

X: I disagree here… We do not call fixing defects firefighting, with vulnerabilities you're right though… Looking at the loops, I think, both are plausible, both may lead to problems, but while the first [defects] is normal in agile methods, the latter [vulnerabilities] should not occur. [**Quote 23**]

Firefighting (Pressure), Defects, Vulnerabilities (Defects & Vulnerabilities)

[This was changed for the final model. Afterwards the discussion continued…]

X: That loop is correct. But the escalation depends on the attacker and his target. [**Quote 32**] A determined state actor will not stop attacking you because of one unsuccessful attempt. Such an attacker will escalate no matter what happens. Somebody who is not interested in specifically targeting you will be much less escalating.

Adversary Strategy, Escalation, state actor (Adversary Behaviour & Cyber Attacks)

Interviewer: Yes, I see. I will emphasise that when talking about it. And if we model that quantitatively one can play with the numbers there… So, let's come to the last loop. You know that mechanism by now: It basically means that your people are firefighting incidents which is why they cannot do their normal work anymore.

Firefighting (Pressure), Incident (Adversary Behaviour & Cyber Attacks)

X: Yes, I know that mechanism and we have discussed it. It's perfectly valid… We would not accept that but that may be our approach. When a team cannot handle its work, you want to cut the pressure loop. We take people from different teams to create new teams to support the stressed DevOps. We sometimes have new teams when DevOps are stressed with vulnerabilities, but especially when incidents occur.

Workload (Pressure), Vulnerabilities (Defects & Vulnerabilities), Incident (Adversary Behaviour & Cyber Attacks), Systems Thinking (Security Measure)

Interviewer: Ok, great. That is really interesting. Thanks. Ok, now I explained you the entire diagram, basically the abstract form of the results of my study here. What do you think, be as critical as you can… what is wrong, what am I missing, what do I need to change?

X: No, it's good. I do not miss anything. And you make the story explicit. I would not believe you if you told me that pressure leads to vulnerabilities but saying that stress leads to defects - which we all know - and defects lead to vulnerabilities makes much more sense. I think this is a great way of putting it. And you are perfectly right here: The less time you have, the more mistakes you have, and the more mistakes you have, the more vulnerabilities you have… Yes, I think you have it. I think your abstraction is perfect and your logic makes really sense. Of course, we may argue that you normally should not end up in an escalatory firefighting behaviour because you should rather cut those cycles before it is too late, but I guess not all companies do that and that does not stand against your description… No, I like it, I think it's very good. *[Very interesting summary]*

Pressure (Pressure), Defects, Vulnerabilities (Defects & Vulnerabilities)

Time Pressure (Pressure), Defects, Vulnerabilities (Defects & Vulnerabilities), Systems Thinking (Security Measure)

Firefighting (Pressure), Escalation(Adversary Behaviour & Cyber Attacks)

Interviewer: Great, thanks a lot! Let's do one final thing.

X: Yes.

[The researcher draws the graphs from Figure 6, the Yerkes-Dodson Law, Figure 10, the Adaptation Trap, and Figure 13, the suggested behaviour of the discussed system, on the wall and explains it to the expert. Then he questions the expert to comment about it. In short, the expert fully confirmed all three of those potential developments and underlined that they are even known among him and his colleagues. When leaving the meeting room, the expert asks the researcher about a critical feedback.]

X: After having done so much investigation here, what do you think, what should we do better?

Interviewer: *Laughing…* First of all, I would say that this company is doing a great job. I do not see that this organisation is in any kind of danger or problems. Of course, some of the issues we talked about, I could see here in the organisation, minor things, but I would say you do fine… But if you ask me where I would look at, I would say at the perception of things. In the very first workshop, you were there too, you guys made really clear that agile doesn't know pressure. Later I could find out that indeed, DevOps have pressure. I would even say, of course they have pressure because pressure is normal. But now we could question that… Can agile have pressure? I think this is what I would recommend: Check whether agile has pressure here and make sure to avoid blundering into the dynamics that we have just discussed with the model… What do you think?

X: That is a very good point, thank you. Yes… Among my colleagues throughout many industries, this is a big question. Nobody knows whether we actually apply agile or whether we simply call it like this and do instead something else [**Quote 37**].

Understanding of Situation (Perception)

Both laughing…

II. E Structure Validation through Disconfirmatory Assessment Interviews

Andersen and colleagues have pointed out that "system dynamics requires the intense use of qualitative data and human judgement in all stages of model development" including the validation of a model (2012, p. 255). In system dynamics research, model validation is understood as a gradual, prolonged, and important process of iteratively and incrementally building confidence in the model with each new insight and test (Barlas, 1996; Forrester, & Senge, 1980). Since the model developed for the purpose of this study is qualitative in nature, no quantitative validation (i.e., structure-behaviour-tests and behaviour tests) was applied. Instead the study relied on common techniques of structure validation in system dynamics research (i.e., Structure Verification, Boundary Verification, Unit Consistency), as well as approaches to construct, internal, and external validity and reliability in case study research (Yin, 2014).

One technique applied three times throughout the six month on site is the disconfirmatory assessment interview (hereafter referred to as disconfirmatory interview) which was developed and proposed by Andersen and her colleagues (2012). Disconfirmatory interviews confront participants and experts within a study with the outcomes of the research, request them to challenge the results, and make use of common biases and heuristics, such as anchoring or defence routines, which are, in contrast to normal interviews, particularly valuable when attempting to falsify outcomes. In the end, it is the aim to increase the participants' and experts' confidence in the model, to help them address their practical real-world problem, and to improve the overall validity of the study. Since reality is not objectively defined, but subjectively perceived by individuals, validation in system dynamics, and more broadly, in qualitative research is often understood as "a matter of social discussion" (Barlas & Carpenter, 1990) which is why the disconfirmatory interview is an exceptional tool for assessing the validity of system dynamics models and case studies.

Since the construct, internal, and external validity and reliability of the research have been shown throughout the entirety of the study, this subsection does not aim to repeat this process. Instead, it briefly shows to what extent the use of the disconfirmatory interview within this study resembles and differs from the four examples provided by Andersen and her colleagues in their study (2012). The comparison is done within the two tables (11, 12) below which are based on the study from Andersen et al. (2012).

II. Table 2: Comparison of Interviewing Principles in Disconfirmatory Interviews

Characteristic	Employed Approach
Interviewee	Participants from the workshops, responsible team (similar to Black, 2002).
Technology of delivery	Face-to-face (as Black, 2002).
Types of questions	Unstructured, partly semi-structured (similar to Black, 2002; Luna-Reyes, 2004).
Behaviour Presentation	Only discussion of potential behaviour with the Gatekeeper and the main responsible for software security.
Structure Presentation	Causal diagram supported by feedback stories (similar to Diker, 2003).
Recording Technique	Notes due to Confidentiality.
Data Processing	Notes, discussion of notes with colleagues and experts, comparison with other data and literature as common in qualitative research (see e.g., Merriam, 2009; & Yin, 2014).
Data Analysis	Coding and Categorisation, Explanation Building, Model Refinement

II. Table 3: Comparison of Interviewing Features in Disconfirmatory Interviews

Recommended Approach	Employed Approach
Use boundary objects to structure the interviews	Qualitative system dynamics model was shown via a powerpoint presentation on a large screen to enrich the social conversation with participants. Comments were noted on post its and written down by the recorder in.a note book. Since all of the participants had some limited prior experience in system dynamics , the understanding and discussions were enhanced (similar to Black, 2002; and Diker, 2003).
Anchor respondents with concrete and specific content	Qualitative system dynamics model presented to participants by revealing one feedback loop after another, thereby unfolding the story of secure software development, software vulnerabilities, and external cyber attacks (similar to Black, 2002; and Diker, 2003).
Use the deference effect to focus [participant] on disconfirmation	Participants were explicitly asked to interrupt the unfolding stories when they desired clarification or adjustments. Presenting, discussing and changing the model with domain experts meant testing and improving the validity of the model (similar to Black, 2002; and Diker, 2003).
Organise the interview around the model's structure	Unfolding model structure as device to explore and discuss model with little emphasis on formal analysis (similar to Black, 2002; and Diker, 2003).
Tailor the interview to the audience	Most of the participants had worked on the model during the workshops. Others were domain experts. Limited knowledge in system dynamics was present (similar to Black, 2002; and Diker, 2003).
Individual not group interview	Two individual- and one group interview. In contrast to Andersen et al.'s (2012) recommendation, discussing the model in a group with domain experts who had not taken part in the model conceptualisation revealed several interesting insights (similar to Diker, 2003).
Changes explicitly articulated	Changes not explicitly listed, but explicitly explained to the participants.