Application of Axiomatic Design for Agile Product Development

Erik Puik[1,*], and *Darek* Ceglarek[2]

[1]HU University of Applied Sciences Utrecht, Padualaan 99, 3584 CH Utrecht, the Netherlands
[2]International Digital Laboratory, WMG, University of Warwick, Coventry, CV4 7AL, UK

Abstract. Agile, and iterative, development methods for new product development are gaining in popularity under product engineers; where it initially was just applied for software development, now larger adoption takes place for product development in general. The design rules of agile development are somewhat conflicting with the guidelines of Axiomatic Design. In this paper, it is investigated why this is the case, what can be done about it, and how can the strengths of agile development be combined with Axiomatic Design to optimise methods for product design. It is shown that the methods are indeed advising on different and conflicting strategies, however, by attenuating the agile design rules in the early stage of design, and doing the same for AD in the later stage of design, best of both worlds can be combined.

1 Introduction

New product development (NPD) is getting more complex by the year due to the increased means for global communication, mainly supported by the internet. More development groups are globally active, which increases competition in research and development. Speed is an essence when executing NPD due to this competition. Products that are introduced to the market too late will miss substantial turnover [1]. Developers apply methods for 'Systems Engineering' in the development process, being able to oversee elements of the design process in the context of the whole system [2] (the term 'Engineering Design' is also applied in the US). Traditional system engineering models are applied in NPD to describe the process in a linear way, such as the 'Waterfall-Model', 'PRINCE2', or the 'V-Model'. Since the change of the century, the shortcomings of these models have grown to serious obstacles due to the increasing NPD-dynamics of modern days [3]. Agile and iterative methods, 'Scrum' being the most valued of many variants, have proven the ability to handle the dynamics, however, this tends to come at the cost of rigidity in NPD [4]-[6]. E. g. for the scum methodology, this lack of rigidity is mainly caused by client involvement and lack of visibility over the project outside the iterative 'sprints'. To address rigidity in NPD, there are basically two ways to be applied [7]:

(i) Organise the design, and all its elements, by the application of knowledge (as detailed understanding of the design is enabled by knowledge);
(ii) Test preliminary designs as soon as possible to enforce appearance of errors and address them accordingly (letting physics speak).

Since testing is a central theme in the application of agile product development methods, topic (ii) is well-embedded in virtually all agile methodologies. It is typically topic (i) that is causing the problems when agile development methods are used. As such, Scrum-sprints focus more on detailed issues than the 'design of the whole'. Agile development methods have a tendency to move attention from 'the whole' to smaller, more specific problems that stand between the now and the next demonstrable prototype. This is where Axiomatic Design (AD) is expected to contribute. With its Independence and Information Axioms, it focuses on respectively 'Doing the right things' and 'Doing these things right' [8]. Mainly the first proposition may be considered a valuable addition to most agile development methods, maintaining focus on the 'big picture'. Questions that arise are:

- Can typical errors in the agile development process be described in the context of AD;
- What are the consequences on the Axioms if these errors occur;
- How can AD be applied to prevent these errors from happening.

The paper is organised as follows. Section 2 explains the background of linear and agile methods. Section 3 explains the methodology of investigation, and Section 4 inventories possible errors in the (agile) development process. Finally, Section 5 discusses the findings and draws conclusions.

2 Background

In this background section, traditional linear models are inventoried. These linear models have been preceding the agile models that recently gained in popularity. The most-applied agile method being the Scrum-methodology.

* Corresponding author: erik.puik@hu.nl

2.1 The Waterfall Model

Widely applied models in industry are the 'Waterfall-Model' and the 'V-Model'. Royce, who was the first to report the Waterfall-Model [9] criticised the model in the same article blaming the lack of (old school) process iterations and testing. The Waterfall-Model also forms the basis for the process-model of the 'PRINCE' method that was introduced in 1989 (PRojects IN Controlled Environments). PRINCE'2', was a continued development to enable broader application than PRINCE that was mainly intended for ICT developments.

2.2 The V-Model

The V-Model, also based on the Waterfall-Model, was originally introduced by Boehm [10] and simultaneously developed further in Germany and the US in the second half of the eighties [11], [12]. In the 1991 proceedings for the National Council on Systems Engineering (NCOSE); now INCOSE as of 1995, the V-Model was adopted in the US for modelling of mainly software systems. Like all basic Waterfall-Models and PRINCE2, the V-Model suffers from the problem of 'missing iterations' [3], [9]. This is not as much a problem to accountants and project managers as it is for developers and testers. The most damaging aspect might be the effect that the V-Model effectively discourages user involvement in evaluating the design before arriving at the formal testing stages. By then it is too late to make significant changes to the design. It must be mentioned that the need for sufficient iterations was emphasised when Rook introduced the V-Model, but since the model does not specifically visualise it, unilateral application of the model has become the standard for most industrial applicants. Nevertheless, the V-Model, and in somewhat lesser extent the Waterfall-Model, today are popular systems engineering methods in industry since they meet needs for management. Though the V-model was presented over 30 years ago, discussion is still active and many variations of the model are still being developed [13]-[16].

2.3 Agile Methods for Product Development

Shewhart described in 1939 the 'Plan-Do-Check-Act cycle of continuous improvement' based on the principles of empiricism as induced by Bacon in his 17th century work 'Novum Organum' [17], [18]. The initial Plan-Do-Check-Act was advertised more broadly by Deming who replaced the stage 'Check' by 'Study' to emphasise that the analysis in this stage was to prevail over inspection [19]. The method was optimised in the sixties by respectively Asimow and Mesarovic as the 'Iconic model of the Design Process' [20], [21]. The Iconic model introduces the cycle of Analysis, Synthesis, Evaluation, and Communication. This foundation forms the basis for modern iterative models for iterative project control up to date.

The need for a combination of structure and dynamics in the ICT world has led to further development methods for iterative development. Agile software development methodologies focus upon incremental design and hence a cyclic approach. The aim with these methods is to: (i) make the development process more responsive in changing environments, (ii) pursue functioning software over extensive documentation, (iii) centre individuals and their interactions rather than tools and processes, and (iv) value customer collaboration over customer contract negotiation.

Of great influence are the 'Spiral Model of Software Development' by Boehm [22], the 'Engineering Design Process' by Ertas & Jones [23], HP's 'Product Development Process', the 'Scrum development method' [24], and IBM's 'Rational Unified Process' [25]. All these methods were initially developed to streamline software developments but later-on found their ways for broader application.

Scrum may be considered the most valued form within the family of agile development methodologies. Scrum uses incremental development procedures with an objective to get working software into the hands of the stakeholders as quickly as possible. As such, Scrum puts business value functions into stakeholder possession early on in the software development life cycle. The more traditional process-oriented development methods cannot provide this agile capability; stakeholders typically would not have access to any software produced until far later in the process. This agile performance is provided in a straightforward procedure that enhances focus and communication in an iterative process. Scrum starts with the business case just as one would do with process-oriented development. From this point, it diverges from linear development methods. The customer requirements are inventoried and refined in close cooperation with stakeholders and the project group. The remaining requirements or 'User Stories' are kept in a list known as the 'Backlog'. Cycles or 'Sprints' are initiated from the backlog to address the customer requirements with the objective to produce operating solutions. The solutions should be fully functional, tested, and documented with the ability to be shipped as a finished product, though with limited functionality. Sprints may last from one week to a month and their progression is kept in a 'Burn Down Chart' to feed its status back to the team. A structure of usually brief meetings takes care of extra information exchange within the project team and leads to joint decisions that are supported by the customer as he regularly participates meetings.

Scrum and related agile methods also suffer from drawbacks compared to the traditional methods. It may fail at the following aspects: (i) a drawback according to Highsmith & Cockburn is the fact that an external client has to be actively involved in the project [4]. The client has to be able and available to test the typical monthly releases and to suggest new or modified functionalities, (ii) by applying Scrum, the vision of the client highly influences development. Highsmith & Cockburn also show that if the client does not have a clear sense of the product's direction, the members of the development

team will tend to behave in the same way, and the final product can be significantly different to what is expected. This makes the main strength of Scrum also one of the main weaknesses: client involvement in the development process, and (iii) another potential weakness is the relatively low visibility over the project outside sprints. This makes it difficult to estimate how long a project will take or how much it will cost. In projects with external clients, where bidding is used to determine the contractor for projects, this can be a major drawback.

3 Methodology

The methodology that is applied to investigate the success of agile methods is to compare agile development to good practice in AD.

3.1 Dubakov's model for the analysis of agile development

Dubakov describes the essence of agile software development and presents an interesting view on his weblog that matches particularly well with AD [26]. Though the analysis is born from a perspective of agile software development, they seem to work for general product development too.

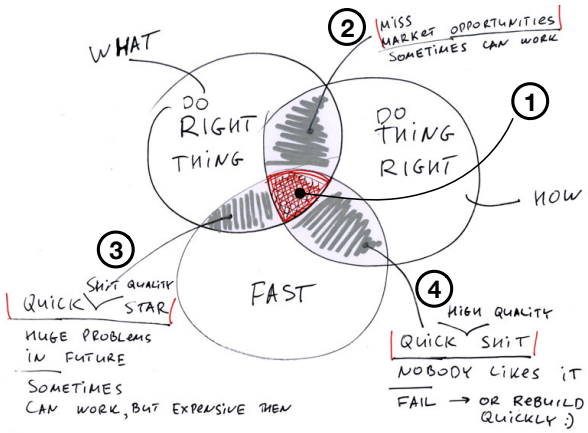

Fig. 1. Three goals of agile development processes as proposed by Dubakov (numbers 1-4 added)

In figure 1, the upper left-hand sphere states 'Do right things' (note that picture states 'thing' but referred text states 'things'). Dubakov explains that these are 'workable things that solve specific problems and solve them well'. This is analogue to satisfaction of the Independence Axiom that forces the definition of: (a) well-chosen Functional Requirements (FRs), (b) matching Design Parameters (DPs) and, (c) an uncoupled or decoupled design. It is only possible to 'Do right things' if the Independence Axiom is satisfied. The right-hand sphere states 'Do things right'. This addresses excellence in execution. From the perspective of AD, it means that the DPs are capable of satisfying the FRs under all circumstances. Doing things right is a process with strong stochastic elements, needing quality in execution, to guarantee that satisfaction of the FRs takes place in all imaginable situations. This means that axiomatic information of the relations between DPs and FRs are eliminated, thus satisfying the Information Axiom. The third element, and here it adds an element that is rather neglected in AD, is the element of development speed, indicated in the lowest sphere of Figure 1. The right things should be executed well, preferably in a small amount of time. It is clear that all three spheres exist in a field of tension.

There are four overlapping areas that have distinctive and characteristic project approaches (shaded areas as indicated and numbered in Figure 1):

(1) Ideal situation, the right things are done well in a short amount of time;
(2) The right things are done in the right way, however, not at the fastest pace;
(3) The right things are done fast, however, they are not done well;
(4) Things are done well and fast, however, they might be the wrong things.

Next, these overlapping areas, further referred to as Project Execution Practices (PEPs) will be evaluated from the perspective of AD.

3.2 Modelling good practice in Axiomatic Design

For the evaluation of the PEPs, a model called the Axiomatic Maturity Diagram (AMD) will be applied. AD prescribes a clear order in which the axioms should be satisfied; start with the Independence Axiom, after that, satisfy the Information Axiom. This design rule was analysed before using the AMD.

Fig. 2. Preferred development path through the Axiomatic Maturity Diagram, as indicated in literature, first moves to the right to satisfy the Independence Axiom. After this, the Information Axiom is satisfied in an upward direction

The AMD is a two-dimensional representation of the Axioms [8], [27]. On the horizontal axis, it shows the progression of the Independence Axiom, from 'No Organisation' to 'Proof of Concept'. On the vertical axis, it shows the progression of the Information Axiom going

from 'Not Robust' to 'Fully Robust'. Product development, in the AMD, will start somewhere at the lower left-hand side and will move diagonally upwards. The design-path, according to 'Good Practice' in AD meets the following demands in that specific order [28]:

- Define FRs and find all relevant DPs to address unrecognised information. Next, if the design matrix is not uncoupled yet, decouple the design matrix to address recognised information (satisfy Axiom 1);
- Match the design ranges and system ranges to guarantee an adequate common range to address axiomatic information (satisfy Axiom 2).

This leads to a preferred path that first moves to the right and then angles upwards, as plotted in Figure 2.

In case of the rather conservative and slow but safe path of the Waterfall-Model, the procedure of following Independence and Information Axioms in that order would be persistent (Figure 3). A slightly riskier path that in practice enhances the development speed of projects is the path of 'Simultaneous Engineering' [8], [29]. This gives the designer more room to start early work on robustness, process technology, and other life cycle elements. It merges the work on Independence and Information Axioms and possibly shortens project lead time.

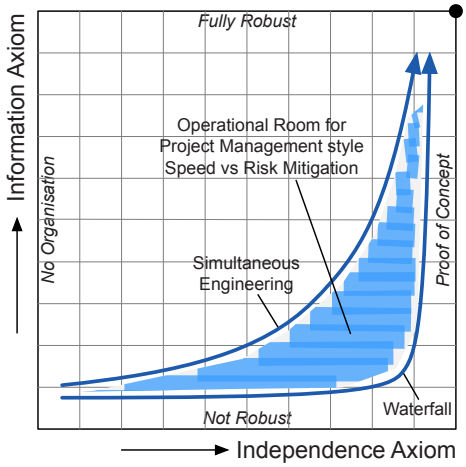

Fig. 3. Depending on the nature of the project, a different strategy may be followed. The right lower curve would represent a waterfall management approach, while the upper would represent the path in case of a simultaneous engineering strategy

Obviously, the safer path is the path proposed by the Waterfall Model, that is also the preferred path in AD. In this situation, the satisfaction of the Independence Axiom will at forehand assure conceptual rigidity of the system design. When this is completed, and the robustness of the system is increased, there is no risk that optimised relations between FRs and DPs need reconsideration. Simultaneous Engineering introduces risks; the fact that the conceptual design is not crystallised may cause conceptual design changes. This means that the FR-DP relations change. If the FR-DP relations were yet optimised, it means that the work for optimisations may indeed have been spent in vain (and work done is lost).

4 Evaluating the PEPs from the perspective of Axiomatic Design

The Project Execution Practices of Figure 1 (overlapping areas 1-4 of paragraph 3.1), will now be investigated using the Axiomatic Maturity Diagram.

(1) The right things are done well in a short amount of time

In this case, little criticism is possible. The product is conceptually strong, robust-engineered, and all that was accomplished in a short amount of time. This case more or less follows the path of simultaneous engineering; conceptual choices happened to be made in a correct manner (path 1 of Figure 4);

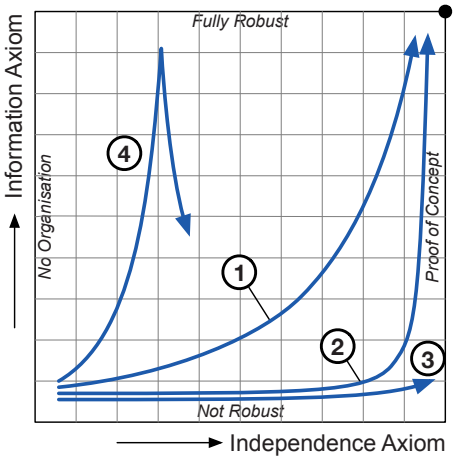

Fig. 4. The four Project Execution Practices and their path though the AMD

(2) The right things are done in the right way, however not at the fastest pace

In the second case, the standard development ways of AD or one of the derivatives of the Waterfall-Model were followed. This development may not be considered to be agile. Market introduction could be late, or later than ideal. In this case, total turnover over the lifecycle of the product may be lower than possible when an agile strategy would have been applied (path 2 of Figure 4);

(3) The right things are done fast, however, they are not done well

This third PEP is recognised because the product, though its concept is smart and well-defined, still does not perform well because the FRs cannot be maintained within their operational areas by the DPs. Customers may be irritated because the product fails. If launched in this state, the service cost may go high, as corrective actions for customers are needed without interruptions. This PEP can be upgraded to PEP 2 by further optimising the design. By doing this, time will slip but the product may still become a 'Good Design' (path 3 of Figure 4);

(4) Things are done well and fast, however, they might be the wrong things

This is the most devastating of all four options. The product has the appearance of a well-engineered product; however, the product is not able to perform well under required circumstances because the underlying concept is structurally inadequate. Problems with the product cannot be fixed easily because it needs a total redesign to correct the bad genes of the design. A lot of work may be spent in vain as the total redesign renounces the conceptual choices of the old design and introduces new DPs to satisfy the FRs. FR-DP relations have to be composed from the ground up (path 4 of Figure 4);.

4.3 Further investigation of PEP 4 and its relation to agile design methodologies

The objective to get working software into the hands of the stakeholders as quickly as possible is an important feature of agile product development methods. This feature comes with strengths and drawbacks:

- The conceptual development of the product is part of multiple sprints. Sprints are initiated from the backlog to address the customer requirements with the objective to produce operating solutions. The solutions should be 'fully functional, tested, and documented with the ability to be shipped as a finished product, though with limited functionality' [30]. This means that relations between FRs and DPs are made robust, while the complete set of FRs is not known yet. As a result, the Information Axiom is addressed before complete satisfaction of the Independence Axiom. As such, product development follows a risky path straight through the middle of the AMD. This is shown in Figure 5, 'End of first sprint';

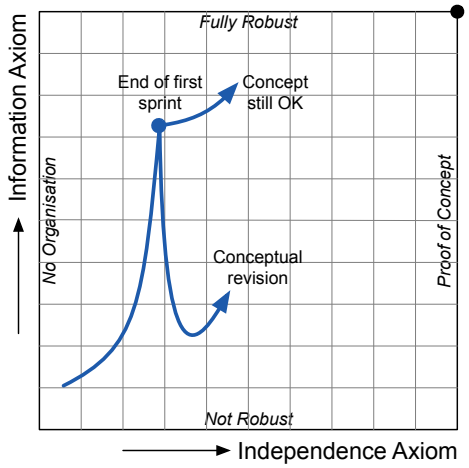

Fig. 5. The Scrum development procedure consists of multiple sprints in which a subset of the FRs are addressed. It is hard to foresee if sprints are able to build further on the result of previous sprints, or that conceptual corrections need to be made. The former situation leads to quick progression of the project. The latter slows the project down and work is spent in vain

- Depending on the conceptual choices made, further FRs may be satisfied in the next sprint without conceptual revision. If this is the case, the next sprint builds further on the previous (Figure 5, 'Concept still OK'). If the concept needs changes in order to satisfy the new FRs, the project status drops in the AMD. In this case, the development gets less efficient; new DPs are needed to satisfy the FRs and earlier optimisations are lost. This is shown in Figure 5, 'Conceptual Revision';
- The efficiency of these agile ways of product design could deliver dramatic results. In an unfortunate situation, the design could need many conceptual revisions. For every single revision, many optimisations of the design would need to be redone causing a lot of work is spent in vain (Figure 6). However, chances of this actually occurring seem moderate from a statistical perspective.

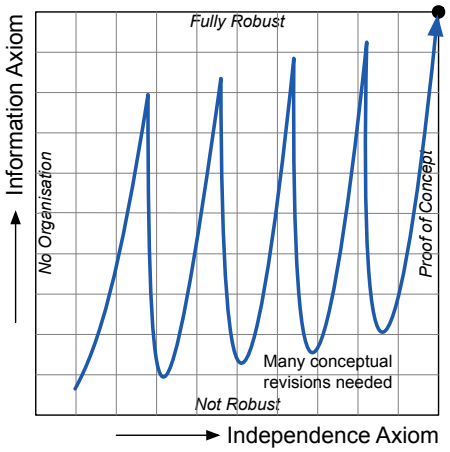

Fig. 6. In a worst-case scenario, many conceptual fixes of the product design would be needed

- A general advantage of agile development methods is that they pull testing towards the present. Earlier investigations have shown that testing and organisation are the most effective way to find hidden complexity ('unknown unknowns') in the design process [1], [8], [27], [31]. This hidden complexity is the main cause for surprises during the design process. Testing is an essential element for agile development, as it substantially increases the chances that these hidden artefacts are found in an early stage of the design process. The sooner hidden problems are found, the sooner they can be addressed, which reduces corrective actions in the conceptual design process. Agile development has great opportunities in this sense;
- Unsuccessful iterations in the design process do not automatically lead to an inefficient development process. As long as iterative cycles are organised as 'safe-fail' experiments, the test will provide positive results; (i) if the test succeeds, it provides for a solution, but (ii) if the test does not succeed, it may provide essential knowledge of the design process. A solid knowledge base of the design and the chosen solutions is essential as the result of the design process never exceeds the state of knowledge of its designers [7].

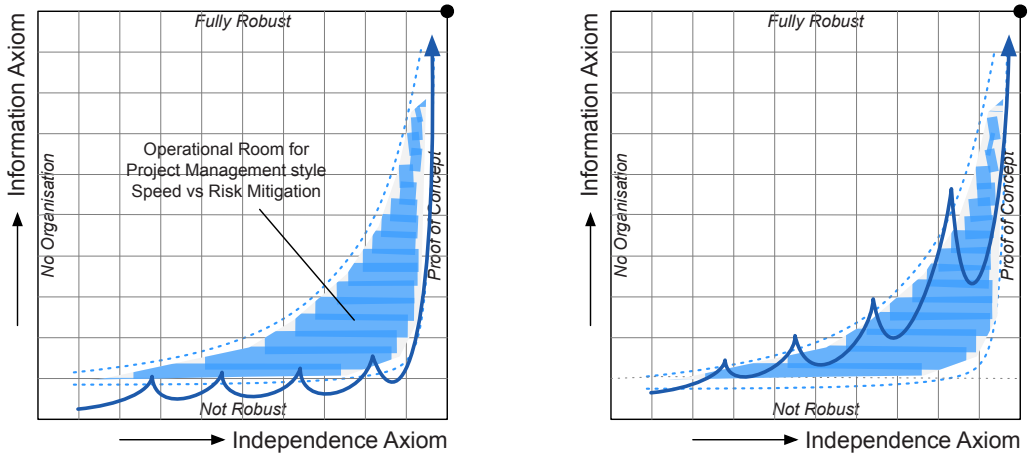

Fig. 7. Alternative development paths that are safer in execution. These paths are not completely conforming the Scrum methodology but approach the safer development order of AD to satisfy the Independence and Information Axioms in that order

5 Discussion

Strengths
Agile development methods, in this case mainly focussing on Scrum and AD appear rather complementary. The rigidity of AD seems superior compared to Scum, however, the power to apply large series of safe-fail tests seems a particular strength of Scrum. The rigidity of AD
and the agile properties of Scrum may also collide; during the development cycles, or sprints, time is limited, and there may be not enough time to spend extensive investigation to decouple the design.

Weaknesses
Scrum uses incremental development procedures 'with an objective to get working software into the hands of the stakeholders as quickly as possible' [30]. This means that every iteration cycle aims to end at the upper side of the AMD (fully robust). Mainly in the beginning of the project, there are substantial uncertainties which may lead to changes in the conceptual design of the product to be developed. This means that many FR-DP-PV relations that have been optimized to become robust are changed and may even be rejected from the final design. In these cases, work for optimisations is spent in vain and work done is lost.

Another problem is the plannability when Scrum is applied. Sprints may be planned as safe-fail processes and as such they can be successful even when the outcome of investigations are negative. Though knowledge development may be considerable, project progression is minimal. It is noted that this could occur in AD as well when the process of zigzagging is not successful and needs to be reconsidered and executed again.

Limitations
Dubakov's 'essence of agile software development' (Figure 1) is not particularly intended for application of traditional linear development methods. The concept however is basically so generic that it should not be a problem to apply it in a broader context. Secondly, the examples in this paper are mainly based on experience and literature. Other agile development methods, as mentioned in the introduction, may have other advantages and limitations. The focus on iterative development cycles though, is generic for most agile development methods and so are the strengths and the drawbacks. Thirdly, this analysis is an analytical approach based on the scientific characterisation of both Scum and AD; practice could work out differently. Therefore, this analysis would benefit from thorough experimental verification. This is not an easy task as such a verification would need many subsequent projects to be executed and monitored under a controlled or at least known environment.

Other considerations
Another finding is based on the analysis of Figure 3 [8]. It shows that it is unwise to spend energy to increase robustness when the system is far from decoupled yet. The chances that FR-DP relations have to be revised are quite large in this stage of development. As the development proceeds, and FR-DP relations start to crystallise, at this point the Independence Axiom is satisfied up to some extent, the chances become significantly smaller. As such, it is wise to stay in the shaded zone of Figure 3, eventually on the upper side. Acting outside this zone increase the chances on harming the FR-DP relations. Unfortunately, this is not what Agile development methods exhort. Using the insights of AD, it would be better to not divert completely from the development path of AD. Which path this should be is difficult to determine based on this study, mainly because the exact location of the higher curve was not investigated yet. Two possible boundaries for such a development path are shown in Figure 7.

6 Acknowledgements

This research was supported by the HU University of Applied Sciences Utrecht, and the project '(G)een Moer An', funded by Stichting Innovatie Alliantie (SIA) in the Netherlands.

References

[1] E. C. N. Puik, "Risk Adjusted Concurrent Development of Microsystems and Reconfigurable Manufacturing Systems," Coventry, 2017.
[2] R. C. Booton and S. Ramo, "The Development of Systems Engineering," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 20, no. 4, pp. 306–310, Jul. 1984.
[3] J. Christie, "The Seductive and Dangerous V-Model," *Testing Experience*, no. 4, pp. 73–77, Dec-2008.
[4] J. Highsmith and A. Cockburn, "Agile software development: the business of innovation," *Computer*, vol. 34, no. 9, pp. 120–127, 2001.
[5] N. F. N. il Ionel, "Critical analysys of the Scrum Project Management Methodology," 2008.
[6] V. Vinekar, C. W. Slinkman, and S. Nerur, "Can Agile and Traditional Systems Development Approaches Coexist? An Ambidextrous View," *Information Systems Management*, vol. 23, no. 3, pp. 31–42, Dec. 2006.
[7] E. C. N. Puik and D. Ceglarek, "The Quality of a Design Will Not Exceed the Knowledge of its Designer; an Analysis Based on Axiomatic Information and the Cynefin Framework," *ICAD2015*, vol. 34, pp. 19–24, 2015.
[8] E. C. N. Puik and D. Ceglarek, "A Different Consideration on Information and Complexity in Axiomatic Design," 1st ed., no. 4, N. P. Suh and A. M. Farid, Eds. 2016, pp. 105–129.
[9] W. W. Royce, "Managing the development of large software systems," *proceedings of IEEE WESCON*, pp. 328–338, Sep. 1970.
[10] B. W. Boehm, "Guidelines for Verifying and Validating Software Requirements and Design Specifications," *IEEE Softw.*, pp. 1–20, 1979.
[11] P. Rook, "Controlling software projects," *Software Engineering Journal*, vol. 1986, no. 1, pp. 7–16, 1986.
[12] J. Friedrich, U. Hammerschall, M. Kuhrmann, and M. Sihling, *Das V-Modell XT*. Berlin/Heidelberg, Germany: Springer, 2009.
[13] P. C. Anitha, D. Savio, and V. S. Mani, "Managing requirements volatility while 'Scrumming' within the V-Model," *2013 IEEE Third International Workshop onEmpirical Requirements Engineering (EmpiRE)*, pp. 17–23, 2013.
[14] M. McHugh, O. Cawley, F. McCaffcry, I. Richardson, and X. Wang, "An agile V-Model for medical device software development to overcome the challenges with plan-driven software development lifecycles," *Software Engineering in Health Care (SEHC), 2013 5th International Workshop on*, pp. 12–19, 2013.
[15] R. Höhn, S. Höppner, M. Broy, A. Rausch, R. Petrasch, S. Biffl, R. Wagner, W. Hesse, and K. Bergner, *Das V-Modell XT*, 1st ed. Berlin/Heidelberg: Springer, 2008.
[16] S. Mathur and S. Malik, "Advancements in the V-Model," *International Journal of Computer Applications*, vol. 1, no. 12, pp. 29–34, 2010.
[17] W. A. Shewhart, *Statistical Method from the Viewpoint of Quality Control*. Mineola, New York: Dover Publications, 1939.
[18] F. Bacon, *Novum organum*. London: Forgotten Books, 1620.
[19] W. E. Deming, *Out of the Crisis*. Cambridge, Massachusetts: MIT Press, 2000.
[20] M. Asimow, *Introduction to design*. Upper Saddle River, New Jersey: Prentice Hall, 1962.
[21] M. D. Mesarovic, *Foundations for a general systems theory*. Cleveland, Ohio, U.S.: Systems Research Center, Case Institute of Technology, 1964.
[22] B. W. Boehm, "A spiral model of software development and enhancement," *Computer*, vol. 21, no. 5, pp. 61–72, 1988.
[23] A. Ertas and J. C. Jones, *The engineering design process*, 1st ed. Hoboken, N.J.: Wiley, 1996.
[24] K. Schwaber, "SCRUM Development Process," no. 11, pp. 117–134, 1997.
[25] P. Kroll and P. Kruchten, *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*. Reading, MA: Addison Wesley, 2003.
[26] M. Dubakov, "The Future of Agile Software Development," *https://www.targetprocess.com/articles/the-future-of-agile-software-development/*.
[27] E. C. N. Puik and D. Ceglarek, "A Theory of Maturity," presented at the 10th International Conference on Axiomatic Design ICAD2014, Lisbon, 2014, 1st ed., pp. 115–120.
[28] N. P. Suh, "Axiomatic Design - Advances and Applications," no. 5, Cambridge, MA: Oxford University Press, New York.Oxford, 2001, Chapter 5, pp. 239-298., 2001.
[29] H. J. Bullinger and J. Warschat, *Concurrent Simultaneous Engineering Systems*. Berlin/Heidelberg, Germany: Springer Science & Business Media, 2012.
[30] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*. 2002.
[31] E. C. N. Puik and D. Ceglarek, "A Review on Information in Design," presented at the 10th International Conference on Axiomatic Design ICAD2014, Lisbon, 2014, 1st ed., pp. 59–64.