Contrasting the Necessary Skills of Leaders in Classical and Agile Software Development

Daniel Staegemann
Otto-von-Guericke University Magdeburg
Magdeburg, Germany *daniel.staegemann@ovgu.de*

Natalie Schröder
Otto-von-Guericke University Magdeburg
Magdeburg, Germany *natalie1.schroeder@ovgu.de*

Christian Daase
Otto-von-Guericke University Magdeburg
Magdeburg, Germany *christian.daase@ovgu.de*

Christian Haertel
Otto-von-Guericke University Magdeburg
Magdeburg, Germany *christian.haertel@ovgu.de*

Matthias Pohl
Otto-von-Guericke University Magdeburg
Magdeburg, Germany *matthias.pohl@ovgu.de*

Robert Häusler
Otto-von-Guericke University Magdeburg
Magdeburg, Germany *robert.haeusler@ovgu.de*

Johannes Hintsch
Otto-von-Guericke University Magdeburg
Magdeburg, Germany *johannes.hintsch@ovgu.de*

Klaus Turowski
Otto-von-Guericke University Magdeburg
Magdeburg, Germany *klaus.turowski@ovgu.de*

Abstract

As a consequence of the necessities of the digital age, agility is becoming more and more important in software development. Consequently, agile change management is increasingly coming into focus and many projects are undergoing a transformation process from classic software development to agile software development. Through this, managers are confronted with new tasks and requirements. To explore the associated effects and needs, this publication examines how managers who have worked in traditional software development apply their skills learned there to agile software development. For this purpose, six interviews with industry experts were conducted and the corresponding results are presented and discussed.

Keywords: Software Development, Agile Transformation, Leadership, Management, Interview.

1. Introduction

With the advance of digitization, software development is changing faster and faster. Projects are becoming increasingly complex and less predictable. Characteristics such as volatility, uncertainty, complexity, and ambivalence characterize the digital age, and as a result, the importance of agility in software development rises [14], [24].

As a result, agile change management is also becoming more and more of a focus and many projects are currently undergoing a transformation process from traditional software development to agile software development. This not only brings uncertainties for the development teams, but the managers are faced with new tasks and requirements as well. Managers who work in classic software development have acquired certain skills over the years and gained a great deal of experience in how best to lead their team members. The classic form of leadership is based on the top-down principle. There is a hierarchical order in which managers give instructions to employees lower in the hierarchy. Responsibility is often delegated upwards. However, as projects become more complex, so do the demands on managers. Reporting and decision-making paths become longer, and decisions are made on the basis of fewer facts. In this way, decisions do not achieve what they are supposed to or are not made quickly enough [8].

Because of this, many organizations are choosing to go down an agile or at least hybrid path [11]. Hybrid forms represent a combination of classic and agile methods. In a study from 2020 [12], in which a total of 642 participants were surveyed, 416 participants stated that they work according to hybrid or selective forms. As the main reason for not using agile approaches across the board, 74 percent said that the framework conditions do not allow for it. The second biggest point, cited by 41 percent, was that the change overwhelms managers. This underlines the significance of executives in the agile transformation.

In an agile organization, autonomous and self-organized teams play a major role, making projects more plannable and adaptable. The teams act according to certain principles. In this way, many problems that exist in classic software development are solved. However, the path to an agile organization is not easy and brings challenges for everyone involved. Agile values and ways of thinking must be internalized and implemented. Above all, managers have to rethink and lead autonomous and self-organized teams from now on. They must embrace agile principles and model them. Leaders would have to recognize that the team with the appropriate knowledge has the power to make decisions.

The big challenge is that leaders need to reshape their role. They need to rethink, use their skills differently, and learn new abilities. Not only are the working methods changing, but also the organizational culture. This causes many concerns and also fears that need to be consciously addressed [17].

By means of expert interviews, the publication at hand explores how the skills of managers differ in agile and traditional software development and to what extent the managers must adapt their previous skills in the course of the agile transformation. Thus, the following research question (RQ) will be answered:

RQ: *How do managers apply their skills from traditional software development to agile software development?*

In order to answer the RQ, the publication is structured as follows. Following this introduction, the concepts of traditional software development and agile software development are briefly outlined in the background section. Afterwards, the methodology, underlying the expert interviews, is described. Subsequently, the results of the interviews and the overall findings are discussed. Finally, a conclusion of the work is provided.

2. Background

In order to discuss classic and agile leadership, it is first important to understand the basics of classic as well as agile software development.

2.1. Traditional Software Development

The term classic software development refers to the application of phase models in software development. The procedure models are thereby plan-driven. Characteristics are,

for example, that a strict time frame is given, and the specifications are fixed at the beginning of the development.

Classical Process Models

Classic process models find their origin in the software crisis in the 70s and 80s. Requirements became more and more complex, and it became increasingly difficult to deliver low-defect software. In the year 1970, Winston Royce [18] presented the first concrete phase model, the waterfall model. He also addresses weak points of the model and proposes improvements. In the following years, the waterfall model was used as a basis for further models. Probably the best known is the V-model developed in 1979 by Barry Boehm [5], which has been further extended several times. The V-model focuses on test and quality management. In the following, the basics of the waterfall model and the V-model will be described.

The Waterfall Model

The waterfall model was first mentioned in 1956 by Herbert Benington [4]. In 1970, it was formally described by Winston Royce. He introduces the idea in the paper "Managing the development of large software systems" [18]. However, the name of the model became established only later.

The waterfall model is characterized by the fact that the phases run sequentially and build on each other. There are various milestones and a resulting document at the end of each phase. Royce first presents a simple form of the waterfall model and then discusses extension possibilities of the model and its risks. The simple form of the waterfall model consists of seven phases. First, there are two phases of requirements specification, followed by a stage of requirements analysis. This is followed by a phase in which the design of the program is defined. Then the program is implemented and subsequently tested. Last follows the phase of start-up and maintenance [18].

A great advantage of the model is its simplicity, which makes it rather easy to understand. Nevertheless, it comes with some disadvantages. Since the phases build on each other, it tempts to define the specifications and requirements at the beginning of the project. As a result, changes at a later stage can hardly be taken into account. In addition, the software is only tested at a very late stage, so that problems are more difficult to rectify and can also cause high costs. Furthermore, executable versions are not available until very late. Therefore, the customer can only provide feedback at a late stage. Another point of criticism is that the first phases are often only based on models and texts, and problems only become visible during implementation. This often leads to time delays. In the extended version, Royce presents proposed solutions for making the model less risky and preventing failure. In doing so, he introduces an iterative idea and adds another design phase [18]. Despite the disadvantages mentioned above, both the simple model and the extended model form the basis for further models in the future, such as the V-model.

The V-Model

The V-Model was described in 1979 by Barry Boehm in "Guidelines for Verifying and Validating Software Requirements and Design Specifications" [5]. It is based on the waterfall model and supplements this with the validation and verification of each phase. As with the waterfall model, each phase is documented.

The V-Model is initiated with a planning and requirements phase in which various acceptance tests validate the requirements. Then, during the product design, the complete system is drafted with a basic architecture and basic design. This results in various acceptance tests for the system. In the subsequent detailed design, the individual components and their interrelationships are defined. In this stage, also corresponding integration tests take place. In the implementation phase, the components are then specified and implemented. The components are tested regularly [5].

Leadership in Classic Process Models

Organizations in which classic software development predominates are characterized by a pyramidal structure. There are few executives at the top, while control and forecasting are

in the foreground here. Classical software development is therefore characterized by the leading role of a project manager, who coordinates the project, makes all important decisions and is the link between the project sponsors and the team members.

2.2. Agile Software Development

Agile software development refers to software development that uses a process model whose elements are based on the values of the so-called Agile Manifesto [3], [9]. The manifesto was published in 2001 and it comprises four guiding values that were postulated as follows:

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

In addition, the authors formulated twelve agile principles [3] that support the understanding of the values:

- "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."
- "Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage."
- "Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale."
- "Business people and developers must work together daily throughout the project."
- "Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done."
- "The most efficient and effective method of conveying information to and within a development team is face-to-face conversation."
- "Working software is the primary measure of progress."
- "Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely."
- "Continuous attention to technical excellence and good design enhances agility."
- "Simplicity--the art of maximizing the amount of work not done--is essential."
- "The best architectures, requirements, and designs emerge from self-organizing teams."
- "At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."

Agile software development is therefore characterized by an incremental and iterative approach, self-organized teams and a close relationship with the customer. As a result, it is possible to react quickly and flexibly to customer requirements.

Agile Process Models

There are many different process models in agile software development. They can be combined or used individually. Scrum, Kanban and Extreme Programming (XP) are presented below. These are among the most popular agile methods and are often combined with each other. They are characterized by certain principles and establish clear rules for software development. In the "15th State of Agile Report" [7], over 1000 people worldwide are surveyed annually about which agile methods they use in their everyday work. 66 percent of the participants stated that they work according to Scrum. Nine percent combine the methods Scrum and Kanban (ScrumBan) and six percent of the respondents combine Scrum and XP. Six percent of the surveyed only use Kanban and one percent only XP.

Scrum

Scrum was developed by Jeff Sutherland and Ken Schwaber in the early 1990s. They were among the authors and the first 17 signatories of the Agile Manifesto [3]. They define Scrum as "a lightweight framework that helps people, teams and organizations generate value through adaptive solutions for complex problems." [21]. Scrum is currently one of

the most successful process models [7], [13]. The method is focused on the tangible progress of the project. The model describes a simple process that is characterized by constant inspection and adaptation. There are three different roles in a Scrum team: a Scrum master, a product owner and several developers. The Scrum master is the facilitator of the team, helping to solve problems and continuously improve collaboration. The product owner represents the ideas and vision of the customer. The teams work in so-called sprints, which are development cycles. Scrum teams are self-organized and have no team leader. There is a product backlog in which the individual components are described in so-called user stories. At the beginning of each sprint, the goals of the respective sprint are defined in sprint planning and the requirements for the next increment of the software are specified. The goals are defined in the form of user stories in the sprint backlog. The team then works on the implementation of the increment for the following weeks. Every working day, there is a daily Scrum meeting to keep track of progress and identify problems. At the end of a sprint, there is a sprint review in which the completed software increment is presented. Customers and users also take part in the sprint review to validate the functionality of the software and provide feedback. Afterwards, another sprint retrospective takes place. Here, the Scrum team reviews how the collaboration can be made even more efficient and effective [21].

There are also certain values that the team follows in order to be able to work together optimally. One is that all team members should have the courage to do the right things and work on hard problems. The other is to focus on the work in the sprint and on the goals. Another value is that everyone is committed to achieving the set goals. The fourth value is to respect each other and the last value is openness [21].

An important feature of Scrum is the self-organization of teams. There is no project manager, the team is responsible for the successful achievement of goals. The team members act on the same level, there is no hierarchy. Classic management tasks such as coordinating and monitoring the completion of work are the responsibility of the Scrum team [15]. By acting as an interface between all members of the project, the Scrum master relieves the team members of small-scale administration and thus enables a self-organized microcosm within the Scrum project. He is responsible for the successful implementation of Scrum and supports the team members in its understanding [21]. The product owner is responsible for maximizing the value of the product through effective product backlog management. He represents the customer's goals and is responsible for ensuring that each team member understands them [21].

Extreme Programming

Extreme programming was developed by Kent Beck, Ward Cunningham and Ron Jeffries. They were among the first 17 signatories of the Agile Manifesto [3]. In 1999, Kent Beck published the ideas in his book: “Extreme Programming explained - Embrace Change”. There, Beck defines XP as a “lightweight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements” [2].

In XP, the focus is on programming. The directly involved actors are the development team, the customer and the product owner. Furthermore, projects are realized in several iterations. Each iteration begins with the iteration planning, in which the team decides together which stories should be implemented in the iteration. The iterations are usually between one and three weeks long. During the iterations, there are daily stand-up meetings to bring each team member up to speed. At the end of an iteration, the result is validated by the customer. During development, frequent testing, pair programming and coding standards play a big role. Beck [2] describes four values to follow in XP:

- **Communication** means that team members communicate with each other on a daily basis to exchange information and questions.
- **Simplicity** means that the team makes the software as simple as possible. It is better to develop simple systems first and extend them later than to develop a complicated system that will not be used to the extent later.
- **Feedback** emphasizes on the one hand the need for continuous testing but also the importance of regular feedback from the customer.

- **Courage** means being willing to try things out and to develop the software conscientiously and sustainably.

In XP, disciplinary supervisors are less important. Beck describes two essential leadership roles. On the one hand, there is the so-called “coach”, who acts in a similar way to the Scrum master. He coordinates the team and the communication with the customer. The coach helps the team solve problems but tries not to hinder the team's independence. He works best by acting indirectly. Second, there is a “big boss”. This person is responsible for the overall project and embodies courage and self-confidence to the team. He supports the development teams and ensures with his assertiveness that the project runs optimally. He gives great importance to communication and explains the necessary steps in detail [2].

Kanban

Kanban originated in 1974 as part of an optimization process in the automotive industry. At the beginning of the 20th century, the ideas were then transferred to software development and finally presented to the public by David Anderson in 2007 [1]. Kanban puts the focus on continuous improvement and is very adaptable since there are only a few guidelines. The method can therefore be integrated easily and quickly into existing processes. Thus, the combination of Kanban with Scrum or XP is straightforward. It serves to improve the workflow and simplify the daily work of the team. Workflows become more flexible, and tasks are processed in small steps. Also with Kanban, similar to Scrum or Extreme Programming, there are various meetings for regular planning and improvement of the process [1]. Further, in Kanban, the work processes are visualized on a so-called “task board” or “Kanban board”. The following six practices describe working with it:

- **Visualize**: This means to make the process visible with the help of a Kanban board.
- **Limit parallel work** (work in progress): Work in progress describes the number of units that are currently being processed. A limit ensures that no new units may be started until the work that is currently in progress is completed. This approach is also known as the “pull method”.
- **Manage the flow of work**: Kaban's goal is to maximize value and minimize cycle times. To achieve this, it is important to review and adjust the flow of work.
- **Make process rules explicit**: Process rules must be understandable and clearly defined. Process rules are for instance work in progress limits or the “Definition of Done”.
- **Implement feedback loops**: Feedback is an important part of any process. Regular reviews are important for effective performance.
- **Improve together, develop experimentally**: Kanban has the goal of continuously improving processes. Changes must therefore be encouraged, and the approach must be continuously adapted.

Kanban requires leadership at all levels of the hierarchy. Therefore, it is even more important that both managers and employees accept the process rules when Kanban is introduced. Thus, managers are expected to lead by example and to be self-reflective. In Kanban, there are no mandatory roles. However, it has been found in practice that it is advantageous to employ a “service request manager” or even a product owner and a “service delivery manager”. The "service request manager" takes on the task of understanding and communicating the needs and expectations of the customer. He or she selects the work units and sets certain requirements for them that the team should fulfill. The “service delivery manager” assists the team in optimizing the approach and resolving issues [1].

2.3. Scaled Approaches

Agile scaling means extending agile approaches to multiple teams and/or projects and/or making the entire company agile. This includes not only pure software development, but also extending agility to the areas of business and project management. In order to illustrate how agile scaling can be implemented, the basics of the Nexus Framework and the Scaled Agile Framework (SAFe) are summarized below.

Nexus

Nexus represents an extension of the Scrum framework. It was primarily developed by Ken Schwaber and published on Scrum.org in 2015. It has been repeatedly updated and further developed in recent years [23].

In general, Nexus is structured very similarly to Scrum. Nexus extends Scrum minimally by using an integration team to help multiple Scrum teams work on a product simultaneously. Nexus consists of three to nine Scrum teams. There is also an integration team, which consists of a product owner, a Scrum master and integration team members. It coaches and coordinates the Scrum teams and makes sure that the framework is implemented correctly. Further, there is a common backlog for the entire product and each Scrum team has its own sprint backlog for each sprint [22].

Scaled Agile Framework

In 2007, Dean Leffingwell first introduced the concept of SAFe. He published his ideas in the book Scaling Software Agility: Best Practices for Large Enterprises. Subsequently, the framework has been continuously developed and improved. SAFe extends agility beyond the team. It specifies processes, roles and structures for different levels of the organization. In doing so, it is used to respond quickly to change, introduce products, adapt to customer and market changes, and prioritize work [20]. In general, SAFe is a very comprehensive framework. It includes various methods, artifacts, meetings, roles, and processes that help scale at different levels of the organization. The framework combines various agile approaches such as Scrum, XP and Kanban. Teams collaborate using an Agile Release Train (ART). It is possible for projects to have multiple ARTs working in parallel.

So that the teams can work together optimally, there are so-called program increments (PI). These consist of three iterations in which increments are programmed. Before each PI, there is a planning session, often lasting several days, in which all members of the ART are present and plan the next three iterations together. At the end of a PI, there is an innovation and planning iteration. This serves the purpose of continuous improvement. On the one hand, the teams have time for further training and problem-solving, and on the other hand, planning for the next PI also takes place in this context. The entire process is carried out in close contact with the customer since it is important to understand the customer and his goals [19].

3. Methodology

To determine, which skills are important in classic and agile leadership and how the skills of classic leaders differ from the skills of agile leaders, expert interviews were conducted. These were focused on the topic of agile software development, how the interviewees perceived the agile transformation, what challenges they had to overcome, and what skills they had to learn or improve in order to make the transformation successful.

The interviews were conducted in the form of partially standardized guided interviews. This means that there was a guideline that specified the questions that were to be answered but there was also room for further explanations by the interviewees, facilitating more comprehensive insights.

Each interview took place in a face-to-face setting via video telephony. Raw interview data were summarized in simultaneous and memory transcripts. In this process, key statements and, in some cases, verbatim statements were noted immediately during the interviews in the form of simultaneous transcripts, and these were subsequently processed in a memory transcript. The interviews were deliberately not transcribed, but merely recorded. This is because the interviews dealt with very personal experiences and feelings, and initially there was no relationship of trust between the interview partner and the person interviewing. According to Gläser and Laudel, when interviews are recorded, there is a risk that the interviewee is biased by the recording and possibly withholds information [10]. Furthermore, the interviews had an explorative purpose. That is, the focus was not on the specific wording, but on the information content [6], [25].

The interviews were analyzed using the qualitative content analysis described by Mayring and Frenzl [16].

3.1. Description of the Sample

Six employees of a large IT service provider were interviewed. The individuals all have experience in traditional software development and are currently working in agile software development as managers. Table 1 gives an overview of the people interviewed. It shows, on the one hand, the positions they held in classic software development and how long they worked in classic software development. On the other hand, the positions that they had or currently have in agile software development and how long they have been working in agile software development are described. The positions are arranged chronologically. The interviewees were all male. Five of the six individuals had worked in traditional software development and have experienced the agile transformation as a manager. One person experienced the agile transformation as a software developer and assumed a leadership role in the subsequent agile software development (IP2).

Table 1. The interviewees

Interview Partner Number	Time in classical software development	Position(s) in classical software development	Time in agile software development	Position(s) in agile software development
IP1	9 years	Software developer, group leader, project manager	2 years	Product owner
IP2	5 years	Software developer	8 years	Software developer Currently: product owner and product manager
IP3	6 years	Software developer, project manager, product owner	7 years	Software developer, Scrum master Currently: product owner and project manager
IP4	6 years	Software developer, project manager	1,5 years	Product owner and product manager
IP5	14 years	Software developer, project manager, team leader	7 months	Product owner
IP6	12 years	Project manager, group leader	2,5 years	Product owner

3.2. Interview Guide

A partially standardized guide was used to structure the interviews. The question formulations and sequence were not binding and could be adapted or supplemented by follow-up questions depending on the course of the interview. This was intended to ensure that the language flowed as naturally as possible [10].

In this context, the interviews first addressed the experiences of the individuals in classic software development (questions 1-4):

- **Q1:** How long did you work in classic projects and which position(s) did you hold?
- **Q2**: What characterized your way of working/your daily work in classic projects?
- **Q3**: What was your leadership style? What leadership values were important to you?
- **Q4**: Which skills were particularly important in your work?

Then, the agile transformation was focused (questions 5-7):

- **Q5**: How did the change from classic to agile come about?
- **Q6**: How did you perceive the change?
- **Q7**: What challenges did you face when switching from classic to agile?

Finally, the work in agile software development was targeted (questions 8-12):

- **Q8**: How long have you been working in agile projects and in which position were/are you working?
- **Q9**: How does your daily work routine differ now from your work in traditional projects?
- **Q10**: How has your work style/daily work routine changed?
- **Q11**: How has your leadership style changed? What leadership values are important to you now?
- **Q12**: Which skills are now particularly important?

4. Findings of the Interviews

In order to evaluate the expert interviews, categories were formed from the interview guide. The following categories were identified:

- **K1**: Leadership in traditional software development
- **K2**: Leadership skills in traditional software development
- **K3**: Challenges in agile transformation
- **K4**: Leadership in agile software development
- **K5**: Leadership skills in agile software development

The interviews were then coded based on the category system and the relevant text passages of the interviews were assigned to the corresponding categories. Based on the category system, the results are summarized in the following.

4.1. Leadership in Classic Software Development

The daily project routine was characterized by defined resources, a fixed time horizon, and a fixed budget (IP1, IP2, IP6). Leadership in traditional software development was described similarly by all interview partners. The project manager manages the projects from start to finish (IP4). Thereby, meeting the deadline was the "be-all and end-all" (IP3). There was a lot of micromanagement and fixed hierarchies. In this, tasks were delegated from the top down almost daily (IP1). Project managers gave instructions to employees, who followed them (IP1). Collaboration was managed by managers and little action was taken as a team (IP2). As a result, employees were very dependent on managers (IP1). Managers assigned tasks without explaining to employees why they should do which tasks (IP1). This often caused problems because the goal and work packages were too large and thus not understood by the employees (IP1).

4.2. Leadership Skills in Traditional Software Development

To be successful as a manager in classic software development, it is important to be close to the product and have a lot of technical knowledge (IP1, IP3, IP4).

In dealing with the team, it is very important to have knowledge of human nature and to respond individually to the employees. Attention was paid to the personal development of the employees (IP1, IP2, IP5). Thus, empathy, meaning a sincere personal interest in the employee (IP1) and friendly interaction with employees, is important (IP2, IP4). Further, managers need to value their employees (IP2).

The manager knew the strengths of each employee and assigned tasks accordingly (IP5), but also tried to prevent unfairness in the distribution of tasks (IP6). Accordingly, it is also important to be pragmatic about what is significant and what was not (IP1). The employees were focused on the manager and committed to her (IP2).

Communication is a highly relevant skill in traditional software development (IP3). Everyone should dare to say everything. There was an open error culture and no finger pointing (IP4, IP6). However, communication also meant making the manager's expectations clear to employees (IP3, IP4, IP6). Managers have to be able to exert pressure and say directly what is bothering them and what is important (IP3, IP4).

4.3. Challenges in the Agile Transformation

The individual experiences of the executives interviewed varied when dealing with the agile transformation and the challenges. The majority of the interviewees see the agile transformation in retrospect as a positive improvement (IP1, IP2, IP3, IP6). Interviewee 4 says that agile procedures are not necessarily always better than classic procedures. He sees advantages in both approaches. Depending on the project, classical elements could be advantageous. Interviewee 5 is currently in the middle of the agile transformation, which for him currently brings more disadvantages than advantages and many challenges. Currently, he would like to return to the classic approach, but if SAFe would work as it is intended, there would be a lot of potential. For him, however, the advantages of the agile approach are not yet usable. Basically, it can be said that all managers were open to the topic of "agile transformation" at the time of the interview and got involved.

A major challenge for many was to understand what agile actually means, how Scrum

or SAFe work, and how to deal with the methodology (IP1, IP2). Transformation takes time and managers need to internalize this. All employees must first understand the methodology (IP2). Learning to describe problems but without describing how to solve them and empowering the team to find solutions themselves is needed (IP1). It is important to highlight not only the goal, but also the added value (IP6). Many found it difficult to let go (IP1, IP2, IP5). They have to accept that they only have a certain sphere of influence in which they can control things (IP2). They had to learn to accept that the team makes decisions differently than they would have made them and that the team itself decides how tasks are distributed (IP1). The agile leader can no longer directly instruct (IP4). The team needs to understand this and learn to embrace the new situation (IP4, IP6). There is a need to learn how to optimize the team. It is important to teach the team to work together, internalize the mindset (IP2, IP3, IP4), and discuss the pros and cons of transformation (IP6). Leaders must learn to say "no" even when the team asks for direct instructions (IP5). The team should motivate each other (IP4). In addition, it was a challenge to keep the team's back toward upper management and to make it clear that the team makes decisions from now on (IP1).

4.4. Leadership in Agile Software Development

The ceremonies and procedures have made everyday life more predictable. The responsibility of the manager decreases; instead, the team shares responsibility with the leadership (IP1, IP6). Leaders are no longer responsible for solving all problems (IP6). In classical software development, managers specified the “what”, the “by when” and the “how”. Today, they are only responsible for the "what" (IP6). Above all, quality awareness and effort estimation are currently coming to the fore (IP3). In addition, the technical depth of leadership has changed. From now on, managers are not leading technically anymore, but through goals and added value (IP1, IP6). Tasks are no longer assigned and fewer direct instructions are given (IP5). When problems arise, appropriate actions are defined together with employees (IP2). Individual attention is paid to the strengths and weaknesses of each employee and care is taken to ensure that each employee can also perform the tasks of the others and that weaknesses are also encouraged (IP3).

4.5. Leadership Skills in Agile Software Development

Transformation takes time and managers must internalize this. All employees must first comprehend the methodology. Patience and understanding must be learned (IP2, IP3). When dealing with the team, it is still necessary to have people skills and empathy. It is important to deal with each employee individually and to make direct statements when necessary (IP1, IP2, IP3). Managers must learn to let go and trust (IP1, IP6). They need to relinquish responsibility and stay out of programming (IP3). Team members should be empowered to take initiative and learn to act in a self-organized way. In the process, leaders must learn to stand back (IP5, IP6). They must be taught to accept the loss of control and the loss of information (IP6).

In addition, leaders must now trust employees to work within the desired quality and solve problems. The manager must learn to stop thinking about the problems themself (IP6). Communication also remains very relevant. Employees need to be listened to and the manager needs to be open to questions and problems (IP2, IP3, IP4, IP6). It is important to encourage and stimulate open communication between employees (IP3). Maintaining a transparent relationship with employees is also considered essential (IP3, IP6). Employees should be made to openly express their opinions (IP4).

4.6. Summarization

When summarizing the expert interviews, it becomes clear that some of the skills from classic software development are used differently in agile software development.

To give an overview of the differences, they are presented in Table 2. Yet, there are also skills that are directly transferable. Here, especially two shared requirements between the approaches become apparent.

Table 2. Differences in the use of skills by managers between classic and agile software development

Ability	Use in classic software development	Use in agile software development
Ability to apply technical knowledge	Technical knowledge is very important because the manager must be close to the product in order to manage it (IP1, IP3, IP4).	The team has the expertise. Managers must relinquish responsibility and stay out of programming (IP3).
Ability to delegate tasks	Tasks are delegated from the top down (IP1). Project managers give instructions to employees, who follow them.	Tasks are not assigned by the manager, the team organizes itself (IP5).
Problem solving	The manager is responsible for solving all problems (IP6).	The leader describes the problems, but without describing how to solve them (IP1). The team is empowered to find solutions themselves (IP1). When problems arise, appropriate measures are defined together with the employees (IP2).
Ability to improve performance	The professional strengths of the employees are used to achieve optimal results as quickly as possible (IP5, IP6).	Individual attention is paid to the strengths and weaknesses of each employee and care is taken to ensure that weaknesses are also encouraged (IP3).
Assertiveness	Managers should be able to exert pressure and say directly what bothers them and what is important (IP3, IP4).	It is still important to make direct announcements, when necessary, but to a lesser extent than in traditional software development (IP2).
Communication skills	Everyone should dare to say everything, there is an open error culture and no finger pointing (IP4, IP6). However, communication also means making clear to employees the expectations that the manager has (IP3, IP4, IP6).	Communication is still very important. In addition, open communication between employees should also be promoted and encouraged (IP3). Employees should be encouraged to express their opinions openly (IP4).

These are on the one hand **empathy**, including honest, personal interest in employees and friendly interaction with each other (IP1, IP2, IP4) as well as valuing the employees (IP2). On the other hand, **knowledge of human nature** is important in both classic and agile leadership. The personal development of employees should be taken into account (IP1, IP2, IP5) and employees should be dealt with individually (IP1, IP2, IP3, IP5). However, overall, it is noticeable that the differences between classic software development and agile software development outweigh the similarities.

5. Conclusion

With the increasing demand for flexibility in software development, agile approaches are growing in importance and prevalence. Besides the related changes to the associated workflow, this naturally also influences the skills that are necessary for leaders to effectively steer their teams. To further explore this topic, the publication at hand focuses on the question of how managers apply their skills from traditional software development to agile software development. For this purpose, semi-structured interviews with industry experts were conducted to draw from their experiences, gathering real-world insights and, thereby, enriching the corresponding scientific discourse. Moreover, practitioners can use the findings as complimentary input when trying to define the design of software development leadership-related training courses or when deciding who should be promoted to a managerial role, based on their personality and skill profile. Yet, while these insights are already valuable, the limited number of interviewees constitutes a limitation that needs to be kept in mind. Therefore, to further increase the significance, in the future, additional interviews should be conducted, possibly also including a female perspective as well as participants from different companies. Moreover, future studies could also be amended by a comprehensive literature review, turning them into a mixed-methods study.

References

1. Anderson, D.J., Carmichael, A.: Die Essenz von Kanban kompakt. dpunkt.verlag, Heidelberg (2017)
2. Beck, K.: eXtreme programming eXplained: Embrace change. Addison-Wesley (1999)
3. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al.: Manifesto for Agile Software

Development (2001)
4. Benington, H.D.: Production of Large Computer Programs. IEEE Annals Hist. Comput. 5, 350–361 (1983)
5. Boehm, B.W.: Verifying and Validating Software Requirements and Design Specifications. IEEE Softw. 1, 75–88 (1984)
6. Bogner, A., Littig, B., Menz, W.: Der Zugang zu den Experten: die Vorbereitung der Erhebung. In: Bogner, A., Littig, B., Menz, W. (eds.) Interviews mit Experten, pp. 27–47. Springer Fachmedien Wiesbaden, Wiesbaden (2014)
7. Digital.ai: 15th State of Agile Report (2021)
8. Eissfeldt, K., Jaeger, C.: So wird Ihr Unternehmen zum wertvollen Arbeitgeber. Springer Fachmedien Wiesbaden, Wiesbaden (2018)
9. Epping, T.: Grundlagen von Kanban. In: Epping, T. (ed.) Kanban für die Softwareentwicklung, vol. 9, pp. 23–52. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
10. Gläser, J., Laudel, G.: Experteninterviews und qualitative Inhaltsanalyse: als Instrumente rekonstruierender Untersuchungen. VS Verlag für Sozialwissenschaften | Springer Fachmedien Wiesbaden GmbH (2009)
11. Jamous, N., Garttan, G., Staegemann, D., Volk, M.: Hybrid Project Management Methods Efficiency in IT Projects. In: Proceedings of the 27th AMCIS (2021)
12. Komus, A., Kuberg, M., Schmidt, S., Rost, L., Koch, C.-P., Bartnick, S., Graf, E., Keller, M., Linkenbach, F., Pieper, C., et al.: Ergebnisbericht: Status Quo (Scaled) Agile 2019/20: 4. Internationale Studie zu Nutzen und Erfolgsfaktoren (skalierter) agiler Ansätze (2020)
13. Kuhrmann, M., Linssen, O.: Welche Vorgehensmodelle nutzt Deutschland? In: PVM (2014)
14. Lenz, U.: Coaching im Kontext der VUCA-Welt: Der Umbruch steht bevor. In: Heller, J. (ed.) Resilienz für die VUCA-Welt, vol. 15, pp. 49–68. Springer Fachmedien Wiesbaden, Wiesbaden (2019)
15. Maigatter, A.: Gut zu wissen: Führung und Scrum-Teams – wie passt das zusammen? In: Wörwag, S., Cloots, A. (eds.) Zukunft der Arbeit – Perspektive Mensch, vol. 20, pp. 303–312. Springer Fachmedien Wiesbaden, Wiesbaden (2018)
16. Mayring, P., Fenzl, T.: Qualitative Inhaltsanalyse. In: Baur, N., Blasius, J. (eds.) Handbuch Methoden der empirischen Sozialforschung, pp. 633–648. Springer Fachmedien Wiesbaden, Wiesbaden (2019)
17. Rosenberg, S.: Organizational Culture Aspects of an Agile Transformation. In: Lassenius, C., Dingsøyr, T., Paasivaara, M. (eds.) Agile Processes in Software Engineering and Extreme Programming, vol. 212, pp. 279–286. Springer International Publishing, Cham (2015)
18. Royce, W.W.: Managing the development of large software systems (1970)
19. Scaled Agile: SAFe Lean-Agile Principles, https://www.scaledagileframework.com/safe-lean-agile-principles/ [13.03.2023] (2021)
20. Scaled Agile: What Is SAFe?, https://scaledagile.com/what-is-safe/ [13.03.2023] (2023)
21. Schwaber, K., Sutherland, J.: The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game (2020)
22. Scrum.org: The Nexus Guide: The Definitive Guide to Scaling Scrum with Nexus, https://www.scrum.org/resources/nexus-guide [13.03.2023] (2021)
23. Scrum.org: Scaling Scrum with Nexus: A framework to help organizations scale Scrum, https://www.scrum.org/resources/scaling-scrum [13.03.2023] (2023)
24. Timmermann, S., Staegemann, D., Volk, M., Pohl, M., Haertel, C., Hintsch, J., Turowski, K.: Facilitating the Decentralisation of Software Development Projects from a Project Management Perspective: A Literature Review. In: Proceedings of the 4th FEMIB, pp. 22–34. SCITEPRESS - Science and Technology Publications (2022)
25. Vogel, D., Funck, B.J.: Immer nur die zweitbeste Lösung? Protokolle als Dokumentationsmethode für qualitative Interviews. Forum Qualitative Sozialforschung / Forum: Qualitative Social Research, Vol 19, No 1 (2018) (2017)