Munir et al. LGURJCSIT 2023

LGU Research Journal of Computer Science & IT

SSN: 2521-0122 (Online)
ISSN: 2519-7991 (Print)

doi: 10.54692/lgurjcsit.2023.0701417

Vol. 7 Issue 1, January – March 2023

An analysis on key Factors of Agile Project Management

Muhammad Saad Munir[1*]
[1]Department of Computer Science and Information Technology

Email: msaadmct@gmail.com

ABSTRACT:

Agile software development is a new method for developing and implementing software. It depends more on ad hoc approaches to planning and control in favor of the more organic processes of teamwork and mutual education. After reviewing several case studies of business initiatives, this research work focuses agile project management. It discusses the historical context of the shift from conventional management's emphasis on top-down supervision and process standardization to agile's emphasis on self-managing teams, with all of the advantages and complications it entails. In this section, you will study the four cornerstones of agile project management: minimal critical specification, self-organizing teams, redundancy, and feedback and learning.

1. INTRODUCTION

Over the past decade, agile software development has gained popularity as a solution for this problem [1] . Several models have emerged over time to better explain the actions of individuals across the many stages of a team's, projects, or business's development. Levels [2]. When it comes to software development projects, Scrum is the most open and accessible method of management. One of the key principles of agile software development is the merging of the development team with the "customers" who stand in for the actual users of the software. The customer and the development team work together to decide which features are most important between iterations. According to [3], the software project manager's job is to ensure that the team is well-coordinated and that everyone's opinions are taken into account.

What makes up a software project is its individual components, all of which work together to produce an end product. In their most basic forms, project management entails the aforementioned three phases: preparation, execution, and monitoring. The high costs and failure rates of such projects continue to interest scholars and practitioners, making the management of software projects a relevant topic despite tremendous advancements in the field.;

Several management activities are required to oversee the various and complex operations that make up a project. The software industry, like other parts of the business world, has been moving towards standardization with the support of codified, generic project management approaches like PRINCE2 (which was established and is championed by the UK government). Although there is agreement on the definition of project management, there is no commonly held theory of project management or agreed-upon methodology for gauging its success [4].

The linear structure and discrete, mechanical perspectives of the systems engineering and quality sciences of the 1950s and 1960s have a considerable impact on current practices in project management. Conventional methods of project management treat requirements gathering, system design, coding, and testing as separate phases. This makes it more challenging to adjust the project plan to account for shifts in requirements or availability. It seems like following the established approach would result in functional hardware. Yet, the software

development process struggles under this context due to its lack of concreteness and consequent lack of acceptance. To characterize a strategy for resolving these issues by applying engineering concepts to software development, the phrase "software engineering" was coined at a landmark NATO meeting in Garmisch-Partenkirchen in 1968. Because of this, most initiatives to solve software development issues have adopted the strategy of treating software development like an engineering process. According to Hoare (1984), the "rise of engineering" and the application of "mathematical evidence" in software development "promise to revolutionize the obscure and error- prone trade of computer programming to meet the highest standards of a contemporary engineering profession." Lehman (1989) also "architected" software in an attempt to lessen risks during the creation phase.

Agile project management differs from the traditional waterfall approach by focusing on short cycles of iterative and incremental delivery of product features and continuous integration of code revisions. The mindset and strategy of upper management must change to accommodate agile project management. Scrum is a framework for managing software development projects. It specifies three roles: the development team, the facilitator, and the product owner. A typical agile team has seven members, as depicted in Fig. 1.

Figure 1: The work environment of an agile development team

The facilitator is in charge of setting up and leading the development team's meetings so that problems can be discussed and solved as they emerge. The product owner is responsible for setting the priorities for the features that will be built. Apart than that, the group should be able to handle itself. Yet, in practice, many companies assign a project manager to aid a product owner in working on requirements and other non-software- related tasks, such as internal and external reporting. Although agile development does not alter the fundamental skills necessary for software development, it does alter the collaborative, coordinated, and communicative character of software projects. When making the switch from waterfall to agile project management, the emphasis needs to move from detailed planning to make choices on the fly. When dealing with complexity and uncertainty, agile development's emphasis on bottom-up, collaborative problem solving within software development teams becomes all the more crucial [5].

The "kick-off" is one of the most common phases in project management. (Besner and Hobbs 2008). The goals, responsibilities, stakeholders, and preliminary plans for a project are all written out and addressed at the introductory meeting. The client acts as the project's "product owner" in Scrum, and it is their job to define the project's "high-level vision and objectives." Also, under agile methods, the team functions as a self-governing entity under the guidance of a single facilitator. Now there's only the team leader and the team members, with no other internal positions to speak of. However, many companies additionally hire a project manager, who is responsible for overseeing the work of many teams on a given project. A product owner is a common example of a stakeholder, but other interested parties, such as customers or even developers from other projects that share technical infrastructure, might also be present. This project's first picture shoot is seen in Figure 2.

Figure 2: beginning a project with a development team, team facilitator, and accountable customer

The length of an iteration is an important component to consider while establishing the project strategy. Iterations should be shorter if there are frequent adjustments in customer demands or technology, and longer if the environment is more stable. An agile team would frequently construct a high-level plan for numerous iterations before creating a more precise plan for the subsequent iteration. At the

beginning of the process, a product owner could be asked to rate the priority of the features that will be produced. Planning poker, for instance, is used to estimate the features, and it stimulates a debate amongst team members about the actions that need to be followed in order to build a feature. The team then commits to what they will be able to produce in the first iteration. The team's "plan" is then the ordered list of features, and throughout the iteration, the employees responsible for implementing those features are allocated to various sprints. How can the first meeting effectively facilitate feedback and lessons learned? We know that teams need to establish common knowledge in a number of domains through studies on shared mental models. The team's collective awareness of the tasks at hand, the state of the art, the competence of its members, and the interactions among them all creates the team's mental models. One technique to enhance consensus on a single mental model is by the use of planning poker.

Team members' abilities may be shown and new information learned through estimation talks about the tasks at hand and the development technologies being utilized [6].

The steps for poker planning are as follows: Each player is dealt a hand of cards with values that ;roughly follow the Fibonacci pattern (0, 1, 2, 3, 5, 8, 13, 20, 40, etc.). Each person assigns themselves a certain amount of time to do each activity based on the card they choose. Members of the team reveal their cards, and the highest and lowest estimators are questioned on their thought processes. If an agreement cannot be reached, the procedure is repeated until a majority or an average of the votes establishes a decision. If there is a wide range of estimates, the work at hand may need to be broken down into more manageable chunks. There have also been studies done on the effectiveness of using planning poker [7]. Last but not least, establishing transparent procedures ensures that everyone has a consistent mental picture of how the team functions. Because to their simplicity and memorability, agile practices may serve as a readily accessible common mental model for teams.

Scientists in the field of organizational learning used the term "reflective practice" to define "the habit of routinely pausing to reflect on the significance to self and others in one's immediate surroundings about what has just occurred." [8]. It provides insight into one's own and others' previous experiences and may be used as a launching pad for future growth [9]. Finding and naming the results of contemplation, investigation, and action is a part of this procedure. It may provide insight into parts of life that have been overlooked thus far. Kerth argues that residents of a community might gain insight into the issues plaguing their neighborhood and be motivated to work together to find answers by participating in a retrospective. With the help of the retrospective, the community may eventually become the "master of its software process." [10]. It is widely believed ;that retrospectives help participants grow intellectually, emotionally, and behaviorally in addition to offering an opportunity to recognize and appreciate past successes. Further, Derby and Larsen contend that teams may benefit from "whole-team learning" by improving their performance by looking back at their prior efforts to see how they could have done things better [11].

The goals of an agile retrospective are to assist the team gather information, gain understanding, and decide on a course of action. (ibid). Many people use techniques like constructing a timeline of key events or merely brainstorming about "what went well" and "what may be better" to collect data. These conclusions are drawn through a variety of analyses, such as fishbone diagrams, data modeling, and prioritization, of the source material. (see Fig.3). Adjustment choices are made and activities to implement those decisions are scheduled for the next iteration based on this information.

Even though retrospectives are a common practice in today's businesses, not much research has been done on the topic. While many articles discuss how to conduct retrospectives, fewer examine their consequences. A study of crucial procedures in R&D businesses found that "learning from post-project audits" was one of the most promising ways for gaining a competitive edge [12].

Kransdorff (1996) argues that participants' faulty memories make postmortems a breeding ground for disagreements. Gathering information as you go throughout the project, such via quick interviews, is one method he proposes for obtaining more neutral data.

Artifacts in physical form are convenient since they can be quickly referred to, annotated, and ignored [13]. It's simpler to control
the flow of data with a physical board than with an electronic system, which is frequently the

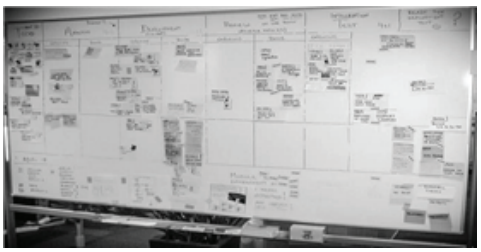

Figure 3: Group of developers conducting a retrospective, organizing ideas generated during a previous session

alternative. Teams may benefit from having a visual representation of the project's progress, the relative significance of tasks, and the degree of product readiness on such boards.

The only steps involved in establishing a visual board are locating a suitable location, deciding on a layout for the cards, and delivering them to the designated area. Try to choose a spot that's convenient for the developmentteam and anyone else who might be following their progress. It might be helpful to have the board next to any visual aids the team is using, such as a burndown4 chart that outlines what tasks need to be completed at this pointin the project. The board should display critical data on the team's current work state and its overall development. Activities in the workplace pass through. To do, analysis, development, review, integration test, and readiness for deployment test are all typical milestones (see Fig. 4). A step in which one developer validates the completion of the job with the consensus of a second developer or an outside party may be worth considering if your team is having trouble with a specific issue, such as developers arguing over whether a certain piece of work is accomplished. Some individuals also use a distinct font color or subheading to draw attention to tasks that might potentially derail the project.

Physical artefacts like the card stand in for symbols of accountability, and it has been observed that physically manipulating objects yields greater insight than using electronic manipulation tools [13].

Physical artefacts like the card stand in for symbols of accountability, and it has been observed that physically manipulating objects yields greater insight than using electronic manipulation tools [13[. Common

issues in a project, such tasks not being done,

Figure 4: Example visual board with areas for tasks "todo," "analysis," "development," "review," "integration test," and "ready for deployment test"

critical jobs not getting done, and too many activities being begun at once, may all be easily seen with a visual board.

Lack of feedback and iterative improvements makes agile project management impossible. Learning and doing go hand in hand because of the agile methodology's emphasis on action rather than preparation. The significance of feedback and learning is further emphasized when the programmer is viewed as an open system that is in constant interaction with its environment. However, because to the complexity and unpredictability of software, problems are notoriously hard to spot until they are virtually resolved .

Due to these obstacles, accurate requirements cannot be specified till a significant portion of the system has been built and put into use. But, unless its purpose is elucidated, the system cannot be constructed. Due of the infinite potential for improvement, this is an ongoing problem. It is more efficient to manage and handle software issues in tiny increments by doing operations like requirements, design, coding, and testing, rather to focusing on a single delivery. The success of the project hinges on how well it handles overlapping and concurrent activities over time. This is accomplished through iterative phases in which providing feedback and learning from past mistakes become second nature. These rules, taken as a whole, provide forth a firm groundwork for a software development project's planning, execution, and monitoring, while leaving room for flexibility in the project's particulars.

2. LITERATURE REVIEW

Traditional project management principles face a number of obstacles, but complexity and unpredictability are particularly significant ones when it comes to the management of software projects. It may be difficult to predict the outcomes of an action in a big project because of

the interplay between the multiple states and actions of the software project and its external elements [14]. An accurate depiction of the technical, organizational, and environmental states that may significantly affect the value of the project's outcome, or of the causal links between them, is simply beyond of the project team's reach in complex software projects.

As Humphrey (1989) pointed out, the most pressing issues in software development are administrative, not technological. As a result, he devised the Capability Maturity Model for Software to standardize and improve the process of creating software (CMM). Both the Capability Maturity Model for Implementation (CMMI) and its predecessor, the Capability Maturity Model (CMM), share the same technical base as software engineers, with an emphasis on predictability and improvement through statistical process management. "Constant improvement" can be anticipated when "the process is under statistical control," as Humphrey (1989) puts it.

The rationalistic traditions of engineering are deeply embedded in the software development and software project management processes, as evidenced by these enlightening quotations from top academics in the software industry. Assemblage line theory as developed by Frederic Winslow Taylor and Henry Ford, as well as Max Weber's study of bureaucracy, can be seen as the foundation for the vast majority of works that came before us. Modern project management methodologies, such as PRINCE2, build on this technical foundation by providing standardized, process-driven alternatives to reactive and flexible methods like Scrum. The whole acronym for Project IN a Networked, Controlled, and Evaluated Environment (PRINCE) is often overlooked. That it doesn't fare as well in the same environment as many (or maybe even most) software development projects is scarcely surprising.

It is projected that for every 25% increase in issue complexity, there is a 100% rise in the complexity of the software solution, and this complexity grows exponentially with scale, causing most of the traditional challenges of software development [15]. Yet another difficulty is that the data required to comprehend most software issues is idea-specific. Most software projects face difficulties that are novel and difficult to formalize, and for which solutions tend to change continuously as programmers acquire a deeper understanding of the problems they face [16].

The fast-paced, highly unpredictable environment, including, for example, market volatility, shifts in client needs, and modifications to project objectives, only serves to heighten the difficulty of the issue and its resolution. As a result, we must recognize that any assumptions or forecasts we make about the future will necessarily include some degree of uncertainty. Management of software projects requires significant caution when extrapolating previous patterns or leaning too much on experience. A future full of surprising changes and unpredictable human behavior is in store, as trends inevitably fade. When a project's inherent uncertainty rises, the team must shift from more conventional techniques based on a defined sequence of activities to those that enable the activities, or even the structure of the project plan, to be rethought and redesigned as the project progresses [17]. Hence, as the complexity and unpredictability of a project rise, managers will need to go beyond conventional risk management, shifting their focus from rigid planning to adaptive problem-solving.

The viewpoint presented in a research is heavily influenced by socio-technical theory as it relates to software project management (Trist 1981). Its basic idea is that organizations are dual-natured social-andtechnical systems, with the interaction between the two being the very essence of a software organization. But, the world is made up of challenges that cannot be reduced to software development approaches, tools, and methodologies, at least from an engineering point of ;view. To put it another way, the technical justification for this worldview prioritizes "objective facts" and global "best practices" above local context and knowledge. Yet, according to socio- technical theory, there may be several optimum solutions to any given issue, since there are often multiple methods to achieve the "joint optimization" of any given technological and human system.

We therefore deny that a high level of formalization is necessary to manage complexity, unpredictability, and change. Keep in mind that (1) a shorter time period is required between planning and execution, (2) an activity's preparation does not supply all the information necessary for its implementation, and (3) creativity and education are required to make sense of the world around us.

Teams are the primary organizational structure for completing software development projects. Individuals with specialized knowledge in design and programming may contribute valuable insights into the inner workings of organizations and their processes in a group environment. However, three challenges characterize the efficiency of software teams[18]. Since the skills required for a software project may be found in a wide variety of people, the first step is for development teams to find out how to communicate, share information, and solve problems efficiently and effectively. Second, software projects can't be completed successfully without formal and informal forms of control, as well as the expertise and understanding to apply both forms at the appropriate periods during the project's execution. Finally, software projects are differentiated by varying degrees of goal ambiguity and coordination problems. Because of this, software development teams need to be adaptable enough to deal with the inevitable challenges that come with working with the unknown. In line with contemporary beliefs, we regard the software team as an integral element of a bigger organizational hierarchy that consists of not only individuals but also other teams and the company as a whole [19]. Having a deeper comprehension of this concept is crucial if we are to succeed in our mission of providing useful guidance for agile software management.

After the advent of agile software development, self-managing software teams have received much acclaim. It is beneficial to organize software development in self-managing teams since it leads to higher levels of productivity, innovation, and employee happiness. To get the most out of people, though, you need to do more than just put them on teams and cross your fingers. Effective management of agile teams requires knowledge of the processes involved in creating and sustaining self-managing teams. Because the project manager is responsible for facilitating distributed leadership (the antithesis of centralized leadership), distributed decision-making, distributed mental models, and a continuous cycle of learning and improvement, leading a self-managing team is more challenging than leading a traditional team. This is a very gradual process. Software development teams are notoriously tough to manage due to the fact that they are often disbanded and reconstituted for each new project. How much time and money can be allocated to the project Tversus how much help is needed [20]. It's unusual for a whole crew to go on from one project to another.

eams that are given the authority to make their own decisions are frequently referred to as "self-managing teams." It's true that self-managing teams are a more recent innovation in software project management, but the idea of self-management has been studied at least since the 1950s, when Eric Trist and Ken Bam forth investigated the ways in which coal miners self-regulated [21]. Software team performance, improvement effort, and creativity have all been proven to benefit greatly from the self-management method. Self-management has been shown to improve job satisfaction, worker loyalty, and attendance [22]. Several academics (Takeuchi and Nonaka, 1986) contend that self-managing teams are crucial to the achievement of innovative goals, particularly in the realm of software development [23]. Cross-training team members to perform several roles increases both functional redundancy and, by extension, the team's flexibility in the case of labor shortages. Despite several studies showing the advantages of self-managing teams, researchers have obtained wildly varied conclusions on the repercussions of such arrangements on productivity, employee turnover, and morale [24]. It has been suggested that self-managing teams are better to traditionally managed teams because they allow decision-making authority to be decentralized to the level of operational challenges and uncertainties, hence improving the efficiency and accuracy with which issues are handled. For the sake of cost reduction, increased productivity, and improved quality, businesses have banded together to create such consortiums. One-personone-vote systems and other democratic ideals won't be enough to create functional self-governing groups, though. [25]. Based on his research, Hackman has identified five overarching criteria that facilitate and encourage self-management.

- Inspiring, well-defined purpose
- A conducive framework for executing units.
- A helpful work environment;
- Ready access to knowledgeable mentors.

and

- Sufficient means

The project manager's role is to ensure that these conditions are met so that the team may become and stay a self-managing agile unit. In order to manage agile projects effectively, you must recognize that team members can take on varying degrees of autonomy. Individuals on an agile team need a great deal of latitude to make decisions on their own. There should be agreement on the team's direction (what activities should be accomplished and how), but each member should have some latitude in selecting how those duties are completed, and the project manager is responsible for making that happen.

The struggle to balance the needs of both individuals and the group is a major barrier to building effective agile teams. If team members are excessively independent and focused on meeting their personal obligations, it might compromise the team's productivity. It's possible that a self-managing team might be more authoritarian in its treatment of its members than is the case under traditional management styles, which would have a dampening effect on morale. The question for a project manager in an agile software development setting is how to find a happy medium between the autonomy of the whole team and the autonomy of its individual members. It is especially challenging to do this in market-driven, agile projects when scope and timelines are often set in stone.

To have a productive team, many experts agree that a strong leader is essential. Although there is a wealth of information out there on many leadership theories. Yet, what leaders should do to improve team performance has received less emphasis in the present research [26]. Some schools of thought on leadership emphasize the importance of mentoring a person more than supervising a team. In this essay, we discuss leadership and decision-making in the context of self-managing teams and examine the pivotal role of team leaders. A recent survey of agile professionals revealed the need of strong leadership in areas like as setting goals, preventing distractions, deciding on a workflow, ensuring sufficient resources, and creating a solid technical foundation [27].

In a self-managing team, everyone is responsible for not only doing their part but also keeping tabs on what they're doing, assessing how well, and thinking of ways to improve it [25]. Due of this, there shouldn't be a monopoly on leadership inside these teams [28] With shared leadership, team empowerment can grow [29]. Leaders and followers alike should take turns at the helm, with authority passing to the person best suited to overcome the challenges at hand ([30]. While the project manager is ultimately responsible for ensuring that all tasks are completed successfully, other team members will contribute when they have expertise that is necessary [31].

At a software company, choices about products and projects can be made on a strategic, tactical, and operational level [32]. In traditional development decision-making is governed by the hierarchical command and control structure, yet, agile development promotes a culture of self-organization and collective decision-making among teams. You'll find a hybrid of the two decision-making styles in most companies. An accurate grasp of the present business process and a deep knowledge of the software product should form the basis for strategic choices in an agile product firm, with the decisions mostly relating to product and release plans. The project management perspective is used to make tactical choices in such organizations, with the overarching goal of figuring out how to best put strategic decisions into action via resource allocation. Decisions regarding how to run an agile business, on the other hand, focus on the details of actually making the product and getting the work done [33]. Since agile teams make strategic choices gradually while delaying crucial tactical and operational decisions as much as feasible, they need to be able to change course quickly in reaction to changing market conditions. Rapid and precise problem resolution is crucial when creating software, and giving the self-managing team ownership of resolving operational issues and uncertainty improves both.

There are many upsides to collaborative problem solving in agile teams, but there are also certain difficulties that might arise. First, compared to the conventional model, where the project manager is accountable for most choices, the shared decision-making strategy is more complex since it incorporates stakeholders with various backgrounds and agendas [34]. Groupthink is perhaps the most well-known issue related to team cohesiveness, but it's not the only one; cohesion has been cited as a cause of inefficient or dysfunctional decision making despite the advantages of shared decision making. Lastly, keep in mind that the team might delegate responsibility to individuals or subgroups within the team, rather than relying on a consensus reached by everyone. Learning who on your team has to weigh in on certain choices might be difficult. As

the standup meeting is concerned with daily coordination and planning, it serves as a crucial platform for collaborative decision making in agile software development. The meeting is meant to be brief and to help with things like better communication, highlighting and promoting swift decision-making, and finding and eliminating roadblocks.

Decisions in the complicated, dynamic, and real-time world of agile software development are made by the team during the daily meeting. The theory of naturalistic decision making (NDM) (Meso et al., 2002) may shed light on the impact of decision making in such a complicated setting when time is of the essence. Experts, according to NDM, can make sound judgments in trying circumstances such limited time, hazy information, and a lack of clarity on the desired outcome without resorting to laborious analysis and weighing of their alternatives. The specialists are able to achieve this because they are able to draw on their prior experiences to spot issues for which they have already devised effective solutions. Specialists employ their knowledge to create mental simulations of the issue at hand, from which they draw conclusions about the best course of action.

As you think about your daily meetings through the lens of NDM, you have to consider certain consequences. The first responsibility of an agile project manager is to ensure that their team members get enough training in both domain-specific competence and collaboration. Second, the agile project manager should ensure that the team is comprised of professionals rather than amateurs, since NDM is dependent on the knowledge and experience of those with advanced training in the field. The agile project manager is responsible for figuring out a path for any team members who are new to the process to progress through the various levels and eventually become experts. Finally, the project manager must ensure the team has formed a shared mental model, in which all members understand their roles and responsibilities, as well as the knowledge and requirements necessary to complete the tasks at hand, if the team is to make good decisions during these sessions [35].

Decisional errors often result from people being too committed to a single course of action [36]. As leaders commit more resources to an ineffective strategy, tensions rise [37]. Considering the complexity and unpredictability of software development projects, this issue is all too prevalent. One study [38] revealed that escalation of commitment occurs in 30–40% of all software projects. According to the results of a number of studies, business leaders often spend more money to defend their prior expenditures [39]. One could expect agile teams to have less rising commitment scenarios than more conventional teams, given that groups can leverage various viewpoints when making choices. However other studies demonstrate that owing to group polarization and conformity demands, rising tendencies are more common and more severe in group decision-making than in individual decision-making [40].

While working on agile projects, it's crucial to keep team meetings from devolving into a forum for justifying past choices [36]. Teams should keep an eye on their internal procedures and also think about who else may join or monitor their meetings. We recommend that persons from outside the team don't regularly engage in team activities like daily meetings, where they can induce team members to feel the need to justify their choices or provide lengthy reports of what they've accomplished. It's important to pay attention to warning indications of escalation, such as members of the team trying to justify the continuance of a particular course of action or providing more extensive and technical explanations of their work since the previous meeting. In addition, the team must address the warning indicators of rising commitment in their retrospective sessions.

Unlike conventional software development practices, agile development places a far greater emphasis on "functioning software" and "individuals and interactions," necessitating a rethinking a rethinking of how information is handled. In the past, developers have mostly concerned themselves with the management of explicit information, such as written lessons learned in knowledge repositories and recorded processes in electronic process manuals. Knowledge is mostly transferred via conversation in agile methodologies [41]. Although CEOs and knowledge officers continue to priorities costly IT systems, measurable databases, and measurement tools, von Krogh et al. point out that these businesses already possess one of the finest mechanisms for knowledge exchange and creation: their own employees. The value of having open and honest dialogues cannot be overstated [42].

Single- and double-loop learning is a popular

theory of learning that emphasizes the need of feedback. The difference between single- and double-loop learning is that the latter is concerned with the values themselves. It's a red flag that an agile team hasn't figured out the root of the issue if they keep trying to find a solution by switching up their methods.

Their issue and is putting single-loop learning to the test. Taking notes and making adjustments to one's behavior in light of new information are two of the most important aspects of learning, according to L43]. Some of the necessary procedures for this to occur include keeping records and evaluating them, setting specific goals, maintaining those goals throughout time, and having faith in the future's potential.

Agile teams may struggle, though, to fully capitalize on educational opportunities [44] found that many groups overlook obvious problems and don't think about how they might enhance their techniques of working together. Not all teams are able to effectively implement changes based on the feedback they receive, despite the fact that they have regular retrospective sessions. When individuals put up sustained effort to resolve problems, but don't see results, they often give up.

Educational challenges will always be there, but achieving success is crucial. Having agreed-upon mental models of the product, tasks, and procedures used by a team has been found to improve performance in a number of studies. Knowledge redundancy, or the creation of overlapping knowledge, is especially important in uncertain contexts where individuals must perform tasks based on priority rather than the competence of team members. We'll take a look at two of the most crucial times for feedback and education in agile development: the beginning of a project and the retrospective that follows each iteration and release.

The table 1 presents a comparison of various research works and their focus on different aspects of software development projects. Process standardization, statistical process management, and complexity management are the most commonly studied areas by the researchers. Solution scalability and solution specificity were the least focused areas in the research works. Additionally, there were some works that did not specifically focus on any of the mentioned areas.

Finally, agile development methodologies have been noted as placing a greater emphasis on

Table 1: Comparative analysis

Research Works	Process Standard-ization	Statistical Process Management	Complexity Management	Solution Scala-bility	Solution Specificity
Hump hrey (1989)	Yes	Yes	Yes	No	No
Wood field (1979)	No	No	Yes	Yes	No
Nerur and Balije pally (2007)	No	No	No	No	Yes
Hackman (1986) et al.	No	No	No	No	No
Morgan (2006)	No	No	No	No	No
Kirkman and Rosen (1999)	No	No	No	No	No
Pearce (2004)	No	No	No	No	No
Hewitt and Walz (2005)	No	No	No	No	No
Bjrnson & Dingsyr (2008)	No	No	No	No	No
Von Krogh et al. (2000)	No	No	No	No	No

"functioning software" and "individuals and interactions," which necessitates a rethinking of how information is handled in software development projects.

3. CONCLUSION

Agile project management principles have the potential to birth new kinds of systems and organi

zations with emergent properties. Nonetheless, companies should proceed with caution when adopting or implementing such ideas and methods. Projects with a wide range of tasks, team members' skill sets, and supporting technologies are well suited to agile management approaches. They work particularly well in creatively oriented rather than formally structured organizations. The takeaway here is that software companies need to take stock of their readiness before embarking on the path to agility.

Managing agile software projects requires a delicate balancing act between extensive planning and iterative improvement. Discipline and an actionable, documentable, and tractable list of activities and contingencies are provided by planning. Adapting to random or unexpected events is made possible through education. They have different management needs and infrastructural projects. When the complexity and unpredictability of a project are low, more time may be spent on detailed planning, but when they are high, more attention must be paid to ongoing learning. Many software companies' newfound willingness to learn is still in its infancy. The numerous high-profile project failures, however, make it clear that established practices in software project management need to be re-examined. Small, co-located projects are where agile project management is seeing the most interest right now. Agile project management, however, has the potential to address some of the most pressing issues plaguing large scale, cross-national endeavors in the future).

REFERECES

[1] T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe, "A decade of agile methodologies: Towards explaining agile software development," *J. Syst. Softw.*, vol. 85, no. 6, pp. 1213–1221, doi: 10.1016/j.jss.2012.02.033. 2012

[2] S. M. Abramson P, Oza N, "'Agile software development methods: a comparative review. In: Dingsøyr T, Dyba',T, Moe NB (eds) Agile software development. Current research and future directions. Springer, Berlin," pp. 31–59, 2010.

[3] W. S. Augustine S, Payne B, Sencindiver F, "Agile project management: steering from the edges," *Commun ACM*, vol. 48, no. 12, pp. 85–89, 2005.

[4] G. G, "NA history of project management models: from pre-models to the standard models. o Title," *Int J Proj Manag.*, vol. 31, no. 5, pp. 663–669, 2013.

[5] P. M. Conboy K, Coyle S, Wang X, "People over process: key challenges in agiledevelopment," *IEEE Softw,* vol. 28, no. 4, pp. 48–57, 2011.

[6] T. E. Fægri, "Adoption of team estimation in a specialist organizational environment," *Lect. Notes Bus. Inf. Process.*, vol. 48 LNBIP, no. 7465, pp. 28–42, doi: 10.1007/978-3-642-13054-0_3. 2010

[7] K. Moløkken-Østvold, N. C. Haugen, and H. C. Benestad, "Using planning poker for combining expert estimates in software projects," *J. Syst. Softw.*, vol. 81, no. 12, pp. 2106–2117, doi: 10.1016/j.jss.2008.03.058. 2008.

[8] T. Dybå, "The Re ective Software Engineer: Re ective Practice," 2014.

[9] J. A. Raelin, "No TitlePublic Reflection as the Basis of Learning," *Manag. Learn*, vol. 32, no. 1, pp. 11–30, 2001.

[10] N. L. Kerth, P*roject retrospectives: a handbook for team reviews*. 2001.

[11] L. D. Derby E, *Agile retrospectives: making good teams great*. 2006.

[12] M. M. Menke, "Managing R&D for Competitive Advantage," *Res. Manag.*, vol. 40, no. 6, pp. 40–42, doi: DOI: 10.1080/08956308.1997.11671169. 1997.

[13] H. Sharp, H. Robinson, J. Segal, and D. Furniss, "The role of story cards and the wall in XP teams: A distributed cognition perspective," Proc. - *Agil. Conf. 2006*, vol. 2006, pp. 65–75, doi: 10.1109/AGILE.2006.56. 2006,

[14] M. T. Pich, C. H. Loch, and A. De Meyer, "On uncertainty, ambiguity, and complexity in project management," *Manage. Sci.*, vol. 48, no. 8, pp. 1008–1023, 2002, doi: 10.1287/mnsc.48.8.1008.163.

[15] S. N. Woodfield, "An Experiment on Unit Increase in Problem Complexity," *IEEE Trans. Softw. Eng.*, vol. SE-5, no. 2, pp. 76–79, 1979, doi: 10.1109/TSE.1979.234162.

[16] S. Nerur and V. G. Balijepally, "Theoretical reflections on agile development methodologies," *Commun. ACM*, vol. 50, no. 3, pp. 79–83, doi: 10.1145/1226736.1226739. 2007.

[17] A. D. E. Meyer, "Institutional Knowledge at Singapore Management University Managing project uncertainty : From variation to chaos," *MIT Sloan Manag. Rev.*, vol. 43, no. (2), pp. 60–67, [Online]. Available: https://ink.library.smu.edu.sg/lkcsb_research/5450/, 2002.

[18] S. Faraj and V. Sambamurthy, "Faraj, Sammaburthy, 2006, *IEEE.pdf*," vol. 53, no. 2, pp. 238–249, 2006.

[19] S. Publications, "Effectiveness Enhancing Work Groups and Teams," vol. 7, no. 3, pp. 77–124, 2013.

[20] Constantine LL, "Work organization: paradigms for project management and organization.," *Commun ACM,* vol. 36, no. 10, pp. 35– 43, 1993.

[21] K. W. Trist, E. L., & Bamforth, "Some Social and Psychological Consequences of the Longwall Method of Coal-Getting.," *Hum. Relations,* vol. 4, no. 1, pp. 3–33, 1951, doi: 10.1177/001872675100400101.

[22] S. Cohen, "What makes teams work: Group effectiveness research from the shop floor to the executive suite," *J. Manage.*, vol. 23, no. 3, pp. 239–290, doi: 10.1016/s0149-2063(97)90034-9. 1997.

[23] P. Hoegl, M., & Parboteeah, "Autonomy and teamwork in innovative projects," *Hum. Resour. Manage.,* vol. 45, no. 1, pp. 67–79, doi: 10.1002/hrm.20092. 2006.

[24] R. A. Guzzo and M. W. Dickson, "Teams in Organizations: Recent Research on Performance and Effectiveness," *Annu. Rev. Psychol.,* vol. 47, pp. 307–338, doi: 10.1146/annurev.psych.47.1.307. 1996.

[25] H. JR, The psychology of self-management in organizations. doi: org/10.1037/10055-003. 1986.

[26] B. B. B. S. B. P. Kozlowski SWJ, "Work groups and teams in organizations In Borman WC, Ilgen DR, Klimoski RJ (ed) Handbook of psychology," *Ind. Organ. psy- chology. Wiley-Blackwell, New York*, vol. 12, pp. 333–375, doi: 10.1002/9781118133880.hop212017. 2003.

[27] T. Dingsøyr and Y. Lindsjørn, "Team performance in agile development teams: Findings from 18 focus groups," L*ect. Notes Bus. Inf. Process.*, vol. 149, no. 7465, pp. 46–60, doi: 10.1007/978-3-642-38314-4_4. 2013,

[28] G. Morgan, "Images of Organization," *Thousand Oaks*, CA Sage Publ., 2006.

[29] B. Kirkman, B. L., & Rosen, "Beyond self-management: Antecedents and consequences of team empowerment," *Acad. Manag.*, vol. 42, no. 1, pp. 58–74., doi: org/10.2307/256874. 1999.

[30] Pearce CL, "The future of leadership: combining vertical and shared leadership to transformknowledge work," *Acad Manag. Exec*, vol. 18, no. 1, pp. 47–57, 2004.

[31] B. Hewitt and D. Walz, "Using shared leadership to foster knowledge sharing in information systems development projects," *Proc. Annu. Hawaii Int. Conf. Syst. Sci*., vol. 00, no. C, p. 256, doi: 10.1109/hicss.2005.666. 2005.

[32] A. Aurum, C. Wohlin, and A. Porter, "Aligning software project decisions: A case study," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 16, no. 6, pp. 795–818, doi: 10.1142/S0218194006003002. 2006.

[33] N. B. Moe, A. Aurum, and T. Dybå, "Challenges of shared decision-making: A multiple case study of agile software development," *Inf. Softw. Technol*., vol. 54, no. 8, pp. 853–865, doi: 10.1016/j.infsof.2011.11.006. 2012.

[34] S. Nerur, R. Mahapatra, and G. Mangalaraj, "Challenges of migrating to agile methodologies," Commun. ACM, vol. 48, no. 5, pp. 72–78, doi: 10.1145/1060710.1060712. 2005.

[35] R. Lipshitz, G. Klein, J. Orasanu, and E. Salas, "Taking stock of naturalistic decision making.," *Decis. Mak. Aviat.*, vol. 352, pp. 3–24, 2015.

[36] V. G. Stray, N. B. Moe, and T. Dybå, "Escalation of commitment: A longitudinal case study of daily meetings," *Lect. Notes Bus. Inf. Process.*, vol. 111 LNBIP, no. 0373, pp. 153–167, doi: 10.1007/978-3-642-30350-0_11. 2012.

[37] B. M. Staw, "Knee-deep in the big muddy: a study of escalating commitment to a chosen course of action," *Organ. Behav. Hum. Perform.*, vol. 16, no. 1, pp. 27–44, 1976, doi: 10.1016/0030-5073(76)90005-2.

[38] M. Keil, J. Mann, and A. Rai, "Why software projects escalate: An empirical analysis and test of four theoretical models," *MIS Q. Manag. Inf. Syst.,* vol. 24, no. 4, pp. 631–664, doi: 10.2307/3250950. 2000.

[39] A. A. Max H. Bazerman, Toni Giuliano, "Escalation of commitment in individual and group decision making," *Organ. Behav. Hum. Perform.*, vol. 33, no. 2, pp. 141–152, 1984.

[40] G. Whyte, "Escalating Commitment in Individual and Group Decision Making: A Prospect Theory Approach," *Organ. Behav. Hum. Decis. Process.,* vol. 54, no. 3, pp. 430–455, 1993.

[41] T. D. Finn Olav Bjørnson, "Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used," I*nf. Softw. Technol.*, vol. 50, no. 11, pp. 1055–1068, 2008.

[42] N. I. Von Krogh G, Ichijo K, E*nabling knowledge creation.* doi: 10.1093/acprof:oso/9780195126167.001.0001. *2000.*

[43] G. S. Lynn, R. B. Skov, and K. D. Abel, "Practices that Support Team Learning and Their Impact on Speed to Market and New Product Success," *J. Prod. Innov. Manag.,* vol. 16, no. 5, pp. 439–454, doi: 10.1111/1540-5885.1650439. 1999.

[44] D. T. Stray VG, Moe NB, *Challenges to teamwork: a multiple case study of two agile teams. 2011*.