The Journal of the Southern Association for Information Systems

Volume 10 | Issue 1 Article 2

2-22-2023

# A Rhetorical Analysis of the Agile Manifesto on its 20th Anniversary

Adarsh Kumar Kakar
*Alabama State University*, akakar@alasu.edu



Recommended Citation
Kakar, A. K. (2023). A Rhetorical Analysis of the Agile Manifesto on its 20th Anniversary. The Journal of the Southern Association for Information Systems, 10, 20-29. https://doi.org/doi:10.17705/3JSIS.00030

**ABSTRACT**

The Agile Software Development (ASD) method is guided by the Agile manifesto which consists of an Agile philosophy and a set of 12 principles. However, despite the indisputable impact of Agile philosophy and principles on software development around the world, the phenomenon of its popularity has not received requisite attention by researchers. In this article we use the rhetorical analysis by focus group of eight industry experts and academics to understand the attractiveness and appeal of ASD to the software development community. We discover that the time was ripe for its introduction with many paradigms and established approaches getting challenged due to rapid pace of technological changes and rise in business uncertainties. The Agile manifesto and its principles tapped into the mood of the moment and perhaps unwittingly through an amalgamation of popular theories of the time from management and manufacturing created a movement generally welcomed by the software developers, but which intrigued many among the software engineering community who were traditionally strong adherents of Taylorism and its principles.

**KEYWORDS**

Agile software development, agile manifesto, agile principles.

**INTRODUCTION**

For many decades, software engineering was focused on heavy-weight approaches aimed at success in developing increasingly complex business applications speedily, at lesser costs and of higher quality. Formal methods based on scientific management principles using a variety of tools and techniques for measurement and standardization of the software process were adopted in the belief that it would result in success in software development activities (Kakar, 2020). These tools and techniques included Structured Systems Analysis and Design Methodology (Eva, 1994), Information Engineering (Martin and Finkelstein, 1981), Unified Software Development Process (Jacobson, Booch and Rumbaugh, 1999) and OPEN (Graham, Henderson-Sellers and Younessi, 1997).

However, in the late 1990s, as disenchantment with the heavy-weight engineering methods grew, suggestions for improvement came from practitioners culminating in the Agile manifesto (Fowler and Highsmith; 2001):

**Manifesto for Agile Software Development**

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

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

That is, while there is value in the items on the right, we value the items on the left more.

The emerging principles (Table 1) from the Agile manifesto and methods such as Extreme programming (Beck and Gamma, 2002), Scrum (Schwaber and Sutherland, 2007), Crystal methodologies (Cockburn, 2004), Dynamic Software Development Method (Stapleton, 1997), Lean Software development (Poppendieck and Poppendieck, 2003) and Feature Driven Development (Palmer and Felsing, 2001) were together labeled as agile software development (Kakar, 2020).

| Serial Number | Agile Principles |
|---|---|
| 1 | Our highest priority is to satisfy the customer through early and continuous delivery of valuable software. |
| 2 | Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. |
| 3 | Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. |

| | |
|---|---|
| 4 | Business people and developers must work together daily throughout the project. |
| 5 | Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done. |
| 6 | The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. |
| 7 | Working software is the primary measure of progress. |
| 8 | Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely. |
| 9 | Continuous attention to technical excellence and good design enhances agility. |
| 10 | Simplicity--the art of maximizing the amount of work not done--is essential. |
| 11 | The best architectures, requirements, and designs emerge from self-organizing teams. |
| 12 | At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. |

Table 1. The 12 Agile Principles

The Agile Manifesto and principles evolving out of agile practices have precedence in the 14 principles of Deming Management Method (Walton, 1988; Deming, 1986; Deming, 2000). However, while the development of the Deming Management Method was a gradual process evolving from over four decades of quality management practices and Deming's own insights from his long and celebrated consulting experience with firms in the United States and Japan (Walton, 1986: 33-34; Yoshida, 1989; Duncan and Van Matre, 1990), the Agile Manifesto and Principles evolved quickly after just a few years of Agile practices.

Despite its quick evolution, this new approach has had a huge impact on how software is developed worldwide (Dyba and Dingsoyr, 2009). The Agile Manifesto caught on quickly with the software development community creating as much impact on the discipline as the Deming Management Method did on the Quality movement. By 2007, 84% of the respondent organizations were using agile methods within their organizations which rose to an impressive 97% by 2018 (Hoda, Salleh and Grundy, 2018).

On its 20th anniversary, we therefore perform a rhetorical analysis of the Agile Manifesto and principles to discover the reasons for its dramatic success. In the analysis we analyze the rhetorical devices used by the authors of ASD to connect with the practitioners and researchers in the software development community. The goal is to understand not only the persuasive powers of the Agile Manifesto but also the merits and demerits of the propounded principles.

## METHOD

According to Oxford Languages Dictionary there are two meanings ascribed to the word "rhetoric " - the art of effective or persuasive speaking or writing, and a language designed to have a persuasive or impressive effect on its audience, but often regarded as lacking in sincerity or meaningful content. We use rhetoric in this article not in the negative sense associated with demagoguery but in the positive sense of effective scientific writing. We use the first meaning of the word and focus on the rhetorical devices used in proclamation of the Agile Manifesto and Principle. A review of literature shows that there are 4 types of rhetorical devices (Rife, 2010):

1. **Logos.** Seeking to convince or persuade via logic and reason. May use statement by authorities, cited facts and statistics
2. **Pathos.** Seeking to convince or persuade by evoking emotion
3. **Ethos.** Seeking to convince and persuade that the authors are a credible source, that they have the necessary expertise, experience, and judgement to know what is right.
4. **Kairos.** Seeking to convince and persuade on an idea whose time has come.

A team of 4 academics and 4 practitioners participated in 10 one-hour online focus group meetings conducted by the author. The author has long experience of conducting focus group discussions. The academics were from Computer Information Systems, Engineering, Management and Marketing disciplines. The college encourages multi-disciplinary research, and the faculty participates willingly on research projects on a quid-pro-quo basis as well as credit received from the university for contribution to the academic community. The 4 practitioners were industry partners of the university who participated in capstone projects for graduate students.

The focus group was initiated to the concept of types of rhetorical devices and the agile methods and principles in the first focus group session. Some academics/ practitioners were already familiar with these concepts and provided their own perspective for the benefit of the focus group. The next eight focus group sessions were a discovery process to find out the reasons for the dramatic success of the agile manifesto through structured brainstorming sessions using the nominal group technique. The summary notes of the deliberation and the tentative group findings were circulated to all group members and provided the basis for the next session. In the last session the members discussed the overall findings and identified which of these 4 rhetorical devices had the maximum impact on the popularity of the Agile manifesto. The findings of the focus group sessions are summarized below in the order of importance of the rhetorical devices as determined by consensus among the focus group members.

**Kairos**

The Agile Manifesto was timely and mirrored "today's turbulent business and technology change" (Cockburn and Highsmith (2001a). There is greater uncertainty and complexity at the workplace than was ever experienced in the past (Kakar, 2018), which included changes in customer demand, intense competition, rapidly changing technologies, and turbulent economic conditions. These changes were difficult to control as they were external to the organization, resulting in 'unpredictability in the inputs, processes, or outputs of work systems' (Wall, Cordery and Clegg, 2002; Wright and Cordery, 1999; Kakar, 2016).

These changes impacted not only software development efficacy but the effectiveness of manufacturing and production in general. Software project failures were rampant, cost and schedule overruns were common. In manufacturing wastage and quality issues received attention as competition became more intense. Inflexible production lines resulting in long production runs, offered little product variety to customers and resulted in product obsolescence. Introduction of new production paradigms were needed. Lean manufacturing in the 1970s popularized by the Toyota Production System and Agile manufacturing in the 1990s began to address these challenges. These methods provided a viable alternative to the Tayloristic methods of mass production.

Against this backdrop the introduction of the Agile Manifesto and principles found fertile grounds for acceptance among the software community. The traditional heavy weight approaches in software development based on the waterfall model (Royce, 1970) which assumed that complex software systems can be built in a sequential, phase-wise manner and where all of the requirements are required to be gathered at the beginning could not cope with changing requirements. Waterfall model equates software development to a production line conveyor belt. Requirements analysts compile the system specifications and send the comprehensive requirements specification document to "software designers" who complete the high level design (system architecture) and low level design including design of database tables, screen design, report design and code specifications. The design documents are then given to the "developers" to implement the code (Szalvay, 2004). However, in reality, most changes in requirements and technology occur within a project's life span. (Abbas, Gravell and Wills, 2008).

As a result, systems were not aligned to the changes taking place in the externa world. Studies of past software projects show that only 9% to 16% are considered on-time and on-budget (Standish Group International, Inc., "Chaos Chronicles", 1994, http://www1.standishgroup.com//sample_research/chaos_1994_1.php). The existing methods of software development did not have a solution to these issues. As a result, the principles of lean and agile manufacturing also found their way into software development resulting in new methods such as Lean Software development in 1992 and Agile method such as Agile methods such as Scrum in 1996, culminating in the Agile Manifesto in 2001. Just as Leanness is usually seen as a precursor for fully agile manufacturing (Narasimhan, Swink and Kim, 2006), Lean software development (Freeman, 1992) is seen as a precursor to the agile manifesto and principles.

In manufacturing lean production is based on four principles: (1) minimize waste; (2) perfect first-time quality; (3) flexible production lines; (4) continuous improvement (Womack, Jones and Roos, 1996), the Lehigh study (Iococca Institute, 1992) included four dimensions of agile manufacturing 1.Enriching the customer; 2. Cooperating to enhance competitiveness; 3. Organizing to master change; 4. Leveraging the impact of people and information

(Goldman et al., 1995; Gunasekharan and Yusuf, 2002). These principles were adopted in the Agile Manifesto (see Table 2 below).

| | Agile Software Development Principles | Lean/ Agile Manufacturing Principles |
|---|---|---|
| 1 | Our highest priority is to **satisfy the customer** through **early and continuous delivery** of valuable software. | Enriching the customer |
| 2 | Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. | Organizing for change; Flexible production lines; Enriching customer |
| 3 | **Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale. | Enriching customer |
| 4 | Business people and developers must work together daily throughout the project. | Cooperation to enhance competitiveness |
| 5 | Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done. | Leveraging the impact of people and information |
| 6 | The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**. | Cooperation to enhance competitiveness, Leveraging the impact of people and information |
| 7 | **Working software** is the primary measure of progress. | Enriching customer; first time quality |
| 8 | Agile processes **promote sustainable development**. The sponsors, developers, and users should be able to **maintain a constant pace indefinitely**. | Leveraging the impact of people and information |
| 9 | Continuous attention to technical excellence and good design enhances agility. | Continuous Improvement |
| 10 | Simplicity--the art of maximizing the amount of work not done--is essential. | Minimize Waste |
| 11 | The best architectures, requirements, and designs **emerge from self-organizing teams**. | Leveraging the impact of people and information; Cooperation to enhance competitiveness |
| 12 | At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. | Continuous improvement: Cooperation to enhance competitiveness |

Table 2. The 12 Agile Principles from Lean/ Agile Manufacturing principles (Kakar, 2020)

**Pathos**

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

The values on the right side represented the yoke of the heavy-weight methods and the left side (in bold) represent the relief. For example, considering the first value "**Individuals and interactions** over processes and tools", while the taylorist method of mass production brough great benefits to the society such as higher high standard of living, it had deleterious impact on the psychological health of the workers. Repetitive work was tiring, boring and dissatisfying (Fraser, 1947; Walker and Guest, 1952). As a result, a new method of work design, Socio-Technical Systems (STS) was proposed. This approach was people focused rather than process focused and soon became popular as an alternative across the world (Christensen, 1993). In Tayloristic work design engineers using time and motion studies decided not only what tasks should be accomplished but also how it should be accomplished. STS is more focused on people issues and less on tools.

In STS, Self-Organizing teams (SET) defined as "as teams of employees who typically perform highly related or interdependent jobs, who are identified and identifiable as a social unit in an organization, and who are given significant authority and responsibility for many aspects of their work, such as planning, scheduling, assigning tasks to members, and making decisions with economic consequences" (Moe, Dingsoyr and Dyba, 2008). Management oversight is light. SET has many advantages over hierarchical structure. "Bringing decision making to the level of teams enables them to respond quickly and appropriately to changing environments" (Fenton-O'Creevy, 1998; Tata and Prasad, 2004). Self-Organization is considered a hallmark of Agile teams (Hoda, 2006). SETs enhance employee motivation and job-satisfaction, and lower absenteeism and employee turnover (Moe, Dingsoyr and Dyba, 2009).

The second value "**Working software** over comprehensive documentation" provides a sense of release due to the idea of less or no documentation. Darwin once said that "A naturalist's life would be a happy one if he had only to observe and never to write." (Horton, 1995). Programmers love coding but following standards, documenting the code with detailed comments, and getting subjected to code inspection is something they generally hate. Visibility into project progress through working products in short iterations is much more tangible than assessing project progress through a reviewing a set of documents. Additionally, work products at the end of shorter iterative cycles is much more motivating than getting the end-product at the end of a long cycle as in the waterfall method. The costs of persistence is lowered and the closure effect phenomenon takes effect. Psychological research shows the nearer one gets to task completion the higher the employee motivation to finish the task (e.g., Katz and Kahn, 1966). The opportunity for closure effect presents itself often in ASD than in Tayloristic methods.

The third value is about "**Customer collaboration** over contract negotiation". This value brings the concept of customer focus which had brought tremendous gains in the marketing discipline to software development. Collaborating with the customers throughout the project cycle to meet their evolving needs is more satisfying than tossing over the Systems Requirement Specification (SRS) after requirements gathering from the customer to the development team who then develop code in to meet the specifications in isolation of the customer. The classic difference between "meeting the specifications" and "fitness for use" is encapsulated in this principle. When employees interact with the beneficiary of their work it motivates them (Grant (2007, 2008b). Co-creation of software builds trust between supplier and customer and helps break down the traditional barriers between them. ASD can be credited to bring the concept of customer focus rather than product focus for the first time in software development.

The fourth value is "**Responding to change** over following a plan". The heavy weight methods of software development promoted the concept of a software factory where productivity was paramount and innovation not so important. It involved assembly line like processes of comprehensive requirement gathering, designing to the minutest details which include coding specifications, breaking down coding tasks to the smallest component, using Gannt charts and productivity metrics to monitor progress and statistical process control techniques to ensure quality. However, in practices the project requirements change during the production cycle itself. Due to heavy planning changes are difficult to implement leading to frustration of all stakeholders. While process focus works well in stabler environments, under uncertain conditions of rapidly evolving customer requirements and

technological changes heavy upfront panning may not be appropriate. Deployment of agile self-organizing teams provides a viable alternative approach (Nerur, Mahapatra and Mangalraj, 2005).

**Logos**

The Agile Manifesto is not a theory expressed as a model of relationships rendering it almost impossible to validate (Kakar, 2020). Therefore, there are not many empirical studies which could examine the manifesto systematically and compare it with Tayloristic methods which it sought to replace and eventually did. As a consequence, there was a lot of to and between the opponents and proponents of Agile methods resulting in nothing mor than confusion. However, a few studies such as Kakar (2012) were able to compare the two paradigms in software development using the Job Characteristics Model (JCM).

Although dated, the JCM is one of the most influential theories in work design. It specifies that there are basically 5 job characteristics (see Table 3), 3 critical psychological states of employees, the need for meaningful work, the need to be responsible for work outcomes and the need for performance feedback and psychosocial outcomes. These critical states are impacted by the 5 job characteristics (see Table 3) and impact personal and work outcomes such as internal work motivation, quality of work, job satisfaction, employee absenteeism and turnover.

Work design models are relevant in assessing software development methods as a method is essentially a systematic way of performing a task or doing work. The 5 characteristics of JCM encapsulate up the typical differences between the 2 methods (see Table 3 below). JCM also suggests that the 5 job characteristics can be combined into a single index Motivation Potential Score (MPS) by combining the scores of jobs on the five dimensions. A recent empirical study found that Agile methods scored higher than plan driven methods on each of the job characteristics and thereby on MPS (Kakar, 2017). Thus, validating from the work design perspective the superiority of the Agile methods over plan-driven methods of software development.

| | Plan-driven methods of software development | Agile methods of software development |
|---|---|---|
| 1 | Skill Variety<br>The degree to which a job requires a variety of different activities in carrying out the work, which involve the use of a number of different skills and talents of the person. | |
| | Specialized Skills and Designated Roles: Designers, Programmers, Testers | Multi-skilled employees, Role Flexibility |
| 2 | Task Identity<br>The degree to which the job requires completion of a "whole" and identifiable piece of work; that is, doing a job from beginning to end with a visible outcome. | |
| | Tasks are project artifacts such as project plan, requirements specification, design documents, code and test reports. | Focus on “whole task” such as developing working products in short iterative cycles. “Whole tasks” contribute to greater task identity. |
| 3 | Task Significance<br>The degree to which the job has a substantial impact on the lives or work of other people | |
| | Simple and narrow tasks such as developing product components have lower task significance | “Whole tasks” such as developing working products that provide gives competitive advantage to customers provide greater task significance |
| 4 | Task Autonomy<br>The degree to which the job provides substantial freedom, independence, and discretion to the individual in scheduling the work and in determining the procedures to be used in carrying it out | |
| | Upfront Planning by Managers/ Supervisors, Task Assignment and Hierarchical Command and Control structure | Self-Organizing Teams, Task Autonomy at team and individual level, Concertive Control |

| | | |
|---|---|---|
| 5 | Feedback<br>The degree to which carrying out the work activities required by the job results in the individual obtaining direct and clear information about the effectiveness of his or her performance | |
| | Hand-offs from one phase to the next, periodic reviews and inspections, project closure meetings provide sporadic feedback | Practices such as Open Workspace, Co-location of development team, Daily Stand up Meetings, Paired Programming, User Representative on the development team, sprint reviews, project retrospectives provide continuous and intense feedback |

Table 3. Differences in SDMs from JCM perspective (adopted from Kakar, 2017)

**Ethos**

All the 17 participants who huddled together for three days at on February 11-13, 2001, at The Lodge at Snowbird ski resort in the Wasatch mountains of Utah, were accomplished software professionals and included representatives from Extreme Programming, SCRUM, DSDM, Adaptive Software Development, Crystal, Feature- Driven Development and Pragmatic Programming (Fowler and Highsmith, 2001). They were in the unanimous view of the authors of this article an authentic, credible, and competent group to proclaim the *Manifesto for Agile Software Development.*

## CONCLUSION

The authors of this study concluded that the greatest credit for the popularity and appeal of the Agile Manifesto goes to the timing of its introduction (Kairos) followed by Pathos, Lagos, and Ethos. At the time the Agile Manifesto was proclaimed, manufacturing paradigms had already deviated significantly from taylorist methods of mass production to adopt lean and agile manufacturing. Parallelly, Lean software development and Scrum in a departure from the heavy weight taylorist and planning focused approaches had already adopted practices of Lean and Agile manufacturing (Kakar and Kakar, 2020). The Agile manifesto reflected these developments (Table 2) and although the authors refer to themselves as "anarchists" (Highsmith, 2001), the manifesto was wittingly or unwittingly a clever package of theories and practices which had attained a level of maturity and acceptance around that time (Kakar, 2014).

The emotion the manifesto evoked (Pathos) from the software development community (the "mushy" aspect (Highsmith, 2001) was next in importance for its attractiveness and popularity among software development community. The sense of release felt by the software development community through work culture and practices (collaborative versus directive), team structure (self-organizing versus hierarchical), management control (concertive versus bureaucratic) was immense. Autonomy is a fundamental human need is rated higher than any other job characteristic by knowledge workers (Deci and Ryan, 1985; Cheney, 1984; Goldstein and Rockart, 1984; Janz et al., 1997). A large number of studies have affirmed the motivational effects of autonomy such as self-confidence, role-breadth self-efficacy and psychological health (Beecham, Baddoo, Hall, Robinson, and Sharp, 2008; Roberts, Hann and Slaughter, 2006; Parker, 1998; Parker, Wall and Jackson, 1997).

At the time the Agile Manifesto was released, employees, especially knowledge workers, were increasingly desiring a fun workplace, a place that is socially engaging rather than oppressive and tasks creative rather than routine. A majority of workers listed criteria where organizations "make work fun" as an important factor in their job search (Belkin, 2007). It was also observed that people who have fun at work experience less stress (McGhee, 2000; Miller, 1996), demonstrate lower turnover and absenteeism (Zbar, 1999), and are more energized and motivated (Stern and Borcia, 1999). People who have enjoyment at work get along with others better (Meyer, 1999) and provide better customer service (Berg, 2001). The Agile Manifesto had the motivating potential for delivering the aspirations of software developers.

However, the Agile Manifesto is not bereft of logical fallacies. For most of the first decade since its introduction, Agile methods were seen as restricted to small, co-located development teams and non-critical system development (Conboy, 2009). This was mainly because the principles recommended face to face interactions, self-organizing teams and there was absence of process rigor and quality assurance guidance required for critical projects. Numerous attempts were made over the years by practitioners to overcome these limitations with varying degrees of

success. They include newer methods to address scalability, global agile development, distributed agile development, Agile-DevOps, agile automation, automated testing and continuous integration (see Ebert, Gallardo. Hernantes and Serrano, 2016; Alqudah and Razali, 2016; Dingsøyr and Lassenius, 2016). Yet, overall, the Agile Manifesto (2001) is a masterpiece in rhetoric and a potent transformative force for the software industry.

## CONTRIBUTION AND FUTURE RESEARCH

One of the major contributions of this multidisciplinary research is that although the Agile Manifesto does not explicitly identify the theory/ it is based on, in reality it is an amalgamation of the popular theories at that time from across disciplines. By uncovering the theoretical bases for the Agile manifesto through rhetorical analysis, future research can explain the uniqueness of the agile phenomena. For example, a large number of empirical studies have observed the motivating potential of agile methods of software development (e.g., Cockburn and Highsmith, 2001; Layman, Williams and Cunninghma, 2004; Mannaro, Melis and Marchesi, 2004; Mann and Maurer, 2005; Melnik and Maurer, 2006; Dyba and Dingsoyr, 2008; Kakar and Kakar, 2017). The emotions (pathos) that agile methods evoke among software developers could be applied to explain the high motivation of team members using agile methods.

The study also highlights the potential of rhetorical analysis in information system research. Future research can apply this method in improving software development processes and other phenomenon in information systems. An illustrative list could include engendering successful software change management and improving software development performance through better understanding of shared leadership, self-organization, and virtual teams. Rhetorical analysis does not just look at the logic and the credibility of an approach but also timeliness of introduction and its emotional impacts on stakeholders.

## REFERENCES


1. Abbas, N., Gravell, A. M., & Wills, G. B. (2008, June). Historical roots of agile methods: Where did "Agile thinking" come from? In *International conference on agile processes and extreme programming in software engineering* (pp. 94-103). Springer, Berlin, Heidelberg.
2. Alqudah, M., & Razali, R. (2016). A review of scaling agile methods in large software development. *International Journal on Advanced Science, Engineering and Information Technology*, *6*(6), 828-837.
3. Beecham, S., Baddoo, N., Hall, T., Robinson, H. and Sharp, H. (2008). "Motivation in Software Engineering: A systematic literature review," *Information and Software Technology*, (50:9), pp. 860-878.
4. Belkin, L. (2007). When Whippersnappers and Geezers Collide New York Times 07/26/2007. New York Times (7), pp. 26.
5. Berg, D. H. (2001). The power of a playful spirit at work. The Journal for Quality & Participation, 24, 57-62.
6. Cheney, P. H. (1984). Effects of individual characteristics, organizational factors, and task characteristics on computer programmer productivity and job satisfaction. *Information and Management* (7), pp. 209-214.
7. Cockburn A. and Highsmith J. (2001). "Agile Software Development: The Business of Innovation." Computer 34(9): 120-127.
8. Cockburn, A. (2004). Crystal clear: a human-powered methodology for small teams. Pearson Education.
9. Conboy, K. (2009). "Agility from first principles: reconstructing the concept of agility in information systems development," *Information Systems Research* (20:3), pp. 329–354.
10. Deci, E. L. and Ryan, R. M. (1985). Intrinsic motivation and self-determination in human behavior. New York: Plenum.
11. Deming, W.E. (1986). *Out of the Crisis*, Cambridge, MA, MIT Press.
12. Deming, W.E. (2000). *The New Economics: For Industry, Government, Education*, 2nd ed., MIT Press, Cambridge, MA.
13. Dingsoyr, T., and Dyba, T. (2009). "What do we know about agile software development?" *Software, IEEE* (26:5), pp. 6–9.

14. Dingsøyr, T., & Lassenius, C. (2016). Emerging themes in agile software development: Introduction to the special section on continuous value delivery. *Information and Software Technology*, *77*, 56-60.

15. Dingsøyr, T., Nerur, S., Balijepally, V., and Moe, N. B. (2012). "A decade of agile methodologies: Towards explaining agile software development" *Journal of Systems and Software* (*85:*6), pp. 1213-1221.

16. Duncan, W. J., & Van Matre, J. G. (Guest Eds.). 1990. The Gospel according to Deming: Is it really new? Business Horizons, 33(4): 3-9.

17. Dyba, T., and Dingsøyr, T. (2008). "Empirical Studies of Agile Software Development: A Systematic Review," Information and Software Technology (50:9/10), pp. 833-859.

18. Eva, M. (1994). *SSADM Version 4: A user's guide*. 2 ed. The McGraw-Hill International Series on Software Engineering, ed. D. Ince. 1994, London: McGraw-Hill Book Company.

19. Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software Development*, *9*(8), 28-35.

20. Freeman, P. (1992) "Lean concepts in software engineering," *IPSS-Europe International Conference on Lean Software Development, Stuttgart*, pp. 1-8, 1992.

21. Goldman, S.L., Nagel, R.N. and Preiss, K. (1995). Agile Competitors and Virtual Organizations: Strategies for Enriching the Customer. Van Nostrand Reinhold, New York, NY, USA.

22. Goldstein, D. K. and Rockart, J. F. (1984). *An* examination of work-related correlates of job satisfaction in programmer/analysts. *MIS Quarterly* (8), pp. 103-115.

23. Graham, I., Henderson-Sellers, B. and Younessi, H. (1997), *The OPEN process specificat*ion. The OPEN series, ed. B. Henderson-Sellers. 1997, Harlow, England: Addison-Wesley.

24. Gunasekaran, A. and Yusuf, Y.Y. (2002}. "Agile manufacturing: a taxonomy of strategic and technological imperatives," *International Journal of Production Research* (40:6), pp. 1357–1385.

25. Fowler, M., and Highsmith, J., 2001. "The Agile Manifesto," *Software Development* (9), pp. 28-32.

26. Hoda, R. (2006). Self-organizing agile teams: A grounded theory, *Phd Thesis*, Victoria University of Wellington.

27. Hoda, R., Salleh, N., & Grundy, J. (2018). The rise and evolution of agile software development. *IEEE software*, *35*(5), 58-63.

28. Horton, R. (1995). The rhetoric of research. *BMJ*, *310*(6985), 985-987.

29. Iacocca Institute. (1992). *21 st Century manufacturing strategy*, Lehigh University, Bethlehem, PA.

30. Jacobson, I., Booch, G. and Rumbaugh, J. (1999), *The unified software development process.* The Addison-Wesley Object Technology Series, ed. G. Booch, I. Jacobson, and J. Rumbaugh. Reading, Massachusetts: Addison-Wesley.

*31.* Janz, B. D., Colquitt, J. A. and Noe, R. A. (1997). Knowledge worker team electiveness: The role of autonomy, interdependence, team development, and contextual support variables. *Personnel Psychology* (50), pp. 877–904.

32. Kakar, A. K. (2012). A theory of software development methodologies. In Proceedings of the Southern Association for Information Systems Conference.

33. Kakar, A. K. (2014). Teaching theories underlying agile methods in a systems development course. In *2014 47th Hawaii International Conference on System Sciences* (pp. 4970-4978). IEEE.

34. Kakar, A. K. (2016). Enhancing reflexivity in software development teams: Should we focus on autonomy or interdependence? *J. Inf. Technol. Theory Appl.*, *17*(3), 2.

35. Kakar, A. K. (2017). Investigating the Motivating Potential of Software Development Methods: Insights from a Work Design Perspective. *Pacific Asia Journal of the Association for Information Systems*, *9*(4), 5.

36. Kakar, A. K. (2018). Investigating The Synergistic and Antagonistic Impacts of Outcome Interdependence, Shared Vision and Team Reflexivity on Innovation In Software Development Projects. *International Journal of Innovation Management*, *22*(06), 1850050.
37. Kakar, A. K. (2020). A Theory of Effectiveness of Agile Software Development. In AMCISS (2020) Virtual.
38. Kakar, A. and Kakar, A. (2020). A Brief History of Software Development and Manufacturing. In *Proceedings of the Southern Association for Information Systems Conference* (p. 1).
39. Katz, D., and Kahn, R. L. (1966). *The social psychology of organizations*. New York: Wiley.
40. Layman, L., Williams L. and Cunningham L. (2004). Exploring extreme program- ming in context: An industrial case study, In Agile Development Conference, 2004 (pp. 32-41). IEEE.
41. Mann, C., and Maurer, F. (2005). A case study on the impact of scrum on overtime and customer satisfaction, In Agile Development Conference, Proceedings (pp. 70-79). IEEE.
42. Mannaro, K., Melis, M., and Marchesi, M. (2004). Empirical analysis on the satisfaction of IT employees comparing XP practices with other software development methodologies, in: Extreme Programming and Agile processes in software engineering, 166-174.
43. Mariotti, J. (1999). A company that plays together stays together, Industry Week (248:6), pp. 63.
44. Martin, J. and Finkelstein, C. (1981). *Information Engineering*. Vol 1 and 2. Englewood Cliffs, New Jersey: Prentice Hall.
45. McGhee, P. (2000). The Key to Stress Management. Retention, and Profitability? More Workplaee Fun, HR Focus (77:9), pp. 5-6.
46. Melnik, G. and Maurer, F. (2006). Comparative analysis of job satisfac- tion in agile and non-agile software development teams, in XP, 32-42.
47. Meyer, H. (1999). Fun for everyone, Journal of Business Strategy (20:2), pp. 13-17.
48. Miller, L. (1996). Vacationers lament: `Are we having fun yet?' Wall Street Journal, p. B1.
49. Narasimhan, R., Swink, M. and Kim, S.W. (2006). "Disentangling leanness and agility: an empirical investigation," *Journal of Operations Management* (24:5), pp. 440–457.
50. Nerur, S., Mahapatra, R., and Mangalaraj, G. (2005). "Challenges of migrating to agile methodologies," *Communications of the ACM*, pp. 72–78.
51. Parker, S. K., Wall, T. D. and Jackson, P. R. (1997). That's not my job': Developing flexible employee
52. work orientations. *Academy of Management Journal* (40), pp. 899–929,
53. Parker, S. K. (1998). Role breadth self-efficacy: Relationship with work enrichment and other organizational practices. *Journal of Applied Psychology* (83), pp. 835–852.
54. Poppendieck, M., and Poppendieck, T. (2003). *Lean software development: An agile toolkit*. Addison-Wesley Professional.
55. Rife, M. C. (2010). Ethos, pathos, logos, kairos: Using a rhetorical heuristic to mediate digital-survey recruitment strategies. *IEEE Transactions on Professional Communication*, *53*(3), 260-277.
56. Roberts, J. A., Hann, I. H. and Slaughter, S. A. (2006). "Understanding the motivations, participation, and performance of open source software developers: A longitudinal study of the Apache projects," *Management science* (52:7), pp. 984-999.
57. Royce, W. W. (1970). "Managing the development of large software systems," In *proceedings of IEEE WESCON* (26:8).
58. Schwaber, K. and Beedle, M. (2002). *Agile software development with Scrum*. Agile Software Development. Upper Saddle River, New Jersey: Prentice Hall.

59. Stapleton, J. (1997). DSDM, dynamic systems development method: the method in practice. Cambridge University Press.

60. Szalvay, V. (2004). An introduction to agile software development. *Danube technologies*, *3*.

61. Yoshida, K. 1989. Deming management philosophy: Does it work in the United States as well as in Japan? Columbia Journal of World Business, 24(3): 10-17.

62. Walton, M. 1986. The Deming management method. New York: Putnam.

63. Walton, M. (1988). The Deming Management Method: The Bestselling Classic for Quality Management!. Penguin.

64. Womack, J., and Jones, D. (1996). *Lean Thinking: Banish Waste and Create Wealth in Your Corporation*, Simon & Schuster, New York, NY.

65. Zbar, J.D. (1999). Burnout warning signs, Computerworld (33:27), pp. 46.

About the author(s)

**Adarsh Kumar Kakar:** Adarsh Kumar Kakar is a Ph.D in Management Science (MIS track) with an interest in understanding the processes and methods in innovative software development. He has over 3 decades of experience in the software industry and has worked as consultant for many Fortune 500 companies. He is currently working as an Assistant Professor in Computer Information Systems department at Alabama State University. He has published articles in Computers in Human Behavior, International Journal of Human Computer Studies, Interacting with Computers, Journal of Computer Information Systems, Information and Software Technology and AIS Transactions on Human-Computer Interaction.