The Journal of the Southern Association for Information Systems

Volume 10 | Issue 1 Article 3

2-22-2023

What more can Software Development learn from Agile Manufacturing? Some pointers on the 20th anniversary of the Agile Manifesto

Ashish Kakar
Updated - AIS, ashish.kakar@ttu.edu

Recommended Citation
Kakar, A. (2023). What more can Software Development learn from Agile Manufacturing? Some pointers on the 20th anniversary of the Agile Manifesto. The Journal of the Southern Association for Information Systems, 10, 30-40. https://doi.org/doi:10.17705/3JSIS.00031

What more can Software Development learn from Agile Manufacturing? Some pointers on the 20th anniversary of the Agile Manifesto

Cover Page Footnote

I am grateful to the coauthor oif my paper published in SAIS 2020 conference proceedings entitled "A Brief History of Manufacturing and Software" for his valuable guidance, expertise and insights.

ABSTRACT

The concept of agility originated in manufacturing and was later adopted by the software development discipline. In this article we argue that in the process some important aspects of the agility theory have been either ignored or misinterpreted. A historical review of the evolving paradigms and practices in software development and manufacturing on the 20th anniversary of the Agile Manifesto (2001) suggests that if the ideas and principles underlying agility are faithfully implemented it would lead to significant improvement in the software development process.

Keywords
Software Development, Manufacturing, Agility

INTRODUCTION

Taking a historical perspective on software development can provide useful insights. A review of evolution of software development and manufacturing reveals that although not explicitly stated and well researched there has been significant cross-domain sharing between the two disciplines (Kakar and Kakar, 2020). A case in point is the emergence of agile methods which are based on lean and agile manufacturing principles.

Agile methods represent a paradigm shift from traditional, plan-based approaches to software development (Dyba and Dingsoyr, 2009). Ever since its introduction in 2001, the agile manifesto has spawned new methods of software development. The emerging principles (listed in Table 2) from the Agile manifesto and the new methods of software development such as Extreme programming, Crystal methodologies, Dynamic Software Development Method, Lean Software development and Feature Driven Development were together labeled as Agile Software Development (ASD).

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. 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). Scholars and practitioners are now working to transfer the success of agile software development methods in other functions and domains, However, in their article Rigby, Sutherland and Takeuchi (2016) noted that "Agile has indisputably transformed software development, and many experts believe it is now poised to expand far beyond IT. Ironically, that's where it began — outside of IT. "

This conceptual study is based on the premise that if the roots of ASD become strong its branches and seeds can spread far and wide to other domains and functions. Therefore, while there is a wave of articles and special issues on how the Agile software development methods inspired by the Agile Manifesto (2001) can be applied to other domains, we take a reverse approach in this study and investigate whether the concept of agility which originated in manufacturing (Conboy, 2009) has been applied correctly and comprehensively in the context of software development. Or is there a misinterpretation of agility and scope for further learning and improvement through correct understanding and implementation of the principles and practices of applicable manufacturing paradigms from which ASD is derived. We conduct a cross-domain study of software development and manufacturing to find out.

The study involved conducting a systematic and reflective review of existing literature in agile manufacturing and agile software development. The resources searched included Science Direct, Google Scholar, IEEE Explore and ACM Digital Library. Duplicate articles selected from these databases first were removed. The articles were then shortlisted for their relevance to the study based on the title, and later on the basis of their abstract. The shortlisted articles were then quickly read to validate their relevance to the goal of this study and a final shortlist created for in-depth and reflective review. The findings of the reflective review are detailed below.

Evolution of Software Development and Manufacturing Paradigms

A review of software engineering and management literature shows that the evolution of software development methods remarkably mirrors the evolution in manufacturing methods (Kakar, 2014; Kakar, 2020). Further, investigations reveal that the change software development methods have lagged the change in manufacturing paradigms indicating the source of inspiration for software development methods and practices is manufacturing and not the other way around. While software development is less than a century old, manufacturing began when man first started making tools and implements. It is not surprising therefore to discover that the evolution of software development methods has trailed the evolution in manufacturing methods (Table 1).

Manufacturing Paradigms	Software Development Approaches
Craftmanship (pre-1910s)	Code and Fix (1950s)
Taylorism and Mass Production (1910s)	Plan-driven approaches such as Waterfall or V Model (1970s)
Lean Manufacturing (1970s)	Lean Software Development (1990s)
Agile Manufacturing (1990s)	Agile Software Development (2000s)

Table 1. Evolution of Manufacturing and Software Development paradigms

Craftsmanship and Code-and-fix

"In the 1950s, software developers were more like artists and craftsmen just as producers of physical products were before the industrial revolution." (Hannemyr, 1999). Formal methods of control such as division of labor and productivity norms were not yet developed. Like the crafts there was scope for creativity and independence. Skilled programmers like craftsmen had deep knowledge and understanding of their domain. They developed the software iteratively and fixed the bugs in the code until the user was satisfied. This code-and-fix method survived because software was not that complex and there was no better way for developing software. However, the code-and-fix approach did not last long. As the use of software became ubiquitous and organizations relied on computers for their business operations, this laissez faire approach was replaced with more disciplined methods. By the mid-sixties, management wanted software development to be a managed and controlled process much like other industrial activities (Hanemeyr, 1999).

Taylorism and Waterfall

To accomplish this, the concepts of Charles Babbage, Adam Smith and Frederick Winslow Taylor were applied to software development. Adam Smith (1776) suggested division of labor by breaking down complex jobs into simpler jobs as a way of enhancing performance. Expanding on these ideas Charles Babbage (1835) pointed out the added advantages of job simplification such as the requirement of less skilled and hence cheaper labor. Later, F. W. Taylor (1911) introduced Scientific Management with the aim of controlling every work activity, from the simplest to the most complicated. He applied to workers the ideas Whitney (see Mirsky and Nevins, 1952) earlier used for making interchangeable parts.

Taylor analyzed tasks into their minutest details and arrived at a standardized process; the one best way to do the job, just as Eli Whitney analyzed a musket into its smallest parts and made a machine to manufacture each part (Mirsky and Nevins, 1952). Industrial engineers conducted time and motion studies aimed at increasing specialization and standardization of work. Together the ideas of Whitney, Taylor and Ford (of moving assembly line) ushered in the era of mass production.

As applied to software development (See Table 4), these concepts led to the development of factory like concepts. R. W. Bemer of General Electric (Bemer, 1969) was among its earliest proponents. He suggested that General Electric adopt standardized tools to reduce variability in programmer productivity and keep a database of historical records for management control. M.D. Mellroy of AT & T (Mellroy, 1968) emphasized systematic reusability of code for enhancing productivity. Further new Taylorist approaches such as the waterfall model (Royce, 1970) and its variants gained popularity. These methods promoted strong conformance to plan through upfront requirements gathering and systems design and linear sequential development phases (Melnik and Maurer, 2006; Kakar, 2012).

They encouraged division of labor leading to specialized roles of business analysts, system architects, programmers and testers (Melnik and Maurer, 2006; Kakar, 2017b).
Attempts were made to introduce statistical control in software engineering (Huh, 2001). Efficiency of software development processes were measured through the use of control charts. Models such as CMM (Capability Maturity Model) gained popularity for defining and improving software development processes (Huh, 2001). Upfront planning, defined processes, coding standards, inspections and reviews, productivity metrics and statistical quality control became the norm. Managers not only assigned tasks to the team members but also specified how they should be performed (process) and by when (schedule) they should be completed.

Although a substantial improvement over "code-and-fix" approach, Taylorist methods have issues of addressing customers' real business needs and keeping with the development schedules. Under conditions of rapidly evolving customer needs, the approach of first defining requirements fully and then delivering them to the customer after a long gap did not seem appropriate. With increasing problem complexity, changing scope and requirements, and evolving technologies, developers, over time, came to realize that software development projects using this approach may not accomplish the planned project objectives.

Lean Manufacturing and Lean Software Development

Lean manufacturing originated on the shopfloors of Japanese manufacturers and in particular as a result of innovations at Toyota Motor Corporation resulting from a scarcity of resources and intense domestic competition in the Japanese market for automobiles (Ohno, 1988). 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, 1990). The lean approach focusses on creation of value by elimination of waste represented an alternative model to that of capital-intensive mass production. The innovations included the Kanban method of pull production, the just-in-time (JIT) production system, automated mistake proofing and high levels of participative employee problem-solving.

The positive outcomes of Lean manufacturing principles exemplified by the Toyota Production System in terms of productivity, time-to-market, product quality and customer satisfaction aroused the interest of the software industry (Bemer and Dawson, 2003). Lean principles were first applied to software development in the 90s (Freeman, 1992), well before the Agile principles. Although the universal application of Lean principles to knowledge work like software development is still under debate there is general acceptance that more lean principles could be virtually applied to any domain (Poppendieck and Cusumano, 2012).

Originally, the focus of lean software development was on making software development more efficient by removing 'waste'. Anything which did not add value to the customer was identified as waste such as adding extra functionality or extra documentation. But later the principle of Just-in-Time (JIT) was applied in lean software development practices such as not doing the requirements too far before one is ready to design, not doing design too far before one is ready to code and not doing code until one is almost ready to test. The idea is to perform all these tasks in small batches like the lean concept of "one piece flow". The essential principle underlying this approach is to take our focus off productivity and put it towards time and the workflow by avoiding delays between steps, eliminating large queues, and making work more visible (see Table 3).

	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 the 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 architecture, 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 derived from Lean/ Agile Manufacturing

Agile Manufacturing and Agile Software Development

Leanness is usually seen as a precursor for fully agile manufacturing (Gunasekharan and Yusuf, 2002). Although introduced in 2000s, the roots of Agile principles can be traced to both Lean and Agile manufacturing paradigms introduced in the 1970s and 1990s respectively (Conboy, 2009). Agile manufacturing is a further evolution of production methodology following Lean manufacturing. The term agile manufacturing can be traced back to the publication of the report 21st Century Manufacturing Enterprise Strategy (Iococca Institute, 1992). The origins of the "agility movement" stems from US government concerns that domestic defense manufacturing capability would be diminished following the end of cold war in 1989. The following phenomena underscore the reasons for putting agility at the core of manufacturing strategy for the twenty-first century (Goldman et al., 1995):

1. Increasing market fragmentation
2. Growth in the need to produce to order
3. Shrinking product life cycles
4. Globalization of production
5. Distribution infrastructures which support greater customization

Leanness is usually seen as a precursor for fully agile manufacturing (Gunasekharan and Yusuf, 2002). While 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, 1990), the Lehigh study included four dimensions of agile manufacturing (see Table 2): 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).

While the proposed definition of leanness is the maximization of simplicity, quality and economy, agile manufacturing added flexibility and responsiveness to the definition (Gunasekharan and Yusuf, 2002). Various lean approaches, such as mixed model scheduling and level scheduling (also referred to as heijunka), have been developed for flexible production lines, but they work best under stable demand environments (Hines, Holweg and Rich, 2004). As a result, various researchers and practitioners have favored agile solutions (Goldman et al., 1995).

Agile manufacturing addresses customer demand variability by flexible assemble-to-order systems and creating virtual supply chains (Hines, Holweg and Rich, 2004). Virtual supply chains are independent firms with distinctive core competences which come together to exploit market opportunities and disband when they are no longer valuable to each other. Further, agile manufacturing seeks to achieve competitiveness through rapid response and mass customization. While lean manufacturing methods deliver good quality product to consumers at low prices through removal of waste and excess inventory, agile manufacturing focus on rapidly entering niche markets by developing capabilities to address specific needs of individual customers. Table 2 summarizes the reflection of lean and agile manufacturing principles in the agile manifesto.

ASD practices from Lean Manufacturing	ASD practices from Agile Manufacturing
Minimizing waste (from Poppendieck and Poppendieck, 2003)	Enriching the customer (Beck 1999; Scrum Alliance 2008)
Overproduction: Develop only critical user stories	Co-creation of software with customer
Inventory: Story cards are detailed only for current iteration	Creating a common way to view the system by using the system metaphor
Waiting: Deliver in small increments	Use of user stories – feature descriptions written from the customer perspective
Extra Processing Steps: Code directly from user stories; get verbal clarification directly from customer	Burndown charts – project progress is measured by number of user stories completed
Motion: Have everyone in the same room, customer included	Incremental releases of working products allow functionality to be released to the customer early
Defects: Both developer and customer tests	Leveraging Impact of People and Information (Beck 1999; Scrum Alliance 2008)
Transportation: Work directly with customers	Product Vision
Flexible Production Lines (Beck 1999; Scrum Alliance 2008)	Open Work Space
Iterative evolutionary development	Co-location of development team
Dedicated integration computer; Automated builds	Paired Programming
Multi skilled employees	Cooperating to enhance competitiveness (Beck 1999; Scrum Alliance 2008)
Project Velocity measured by number of user stories completed provides visibility	Daily Stand up Meetings, face-to-face communication promotes tacit knowledge sharing
Practices for first-time quality (Beck 1999; Scrum Alliance 2008)	User representative on the development team
Test driven development	Promoting collective ownership
Working products in each iteration	Concertive rather bureaucratic control
Integrate code frequently	Organizing to master change (Beck 1999; Scrum Alliance 2008)
ASD practices for continuous improvement (Beck 1999; Scrum Alliance 2008)	Self-organizing teams
Sprint Reviews	Making customer available as part of ASD team
Periodic refactoring of existing code	Policy of moving people around
Project retrospectives	Recruiting and developing multi-skilled employees

Table 3. Practices adopted by ASD from Lean/ Agile Manufacturing principles

ASD began as a countermovement to the Taylorist software development processes like the Waterfall Model or the V-Model (Fowler and Highsmith, 2001). There is a sharp contrast between Taylorist and Agile software

development approaches. Taylorist approaches are based on the principle that the first step in a product/ system solution is to comprehensively capture the full set of user requirements to address the business problem. This is followed by architectural and detailed design. Coding or construction is commenced only after confirmation of requirement specification by the customer and completion and approval of architecture/ design. The customer is typically involved at the stage of requirements gathering and the final stage of product acceptance. As a result, the validation of the product happens only at requirement gathering stage and at the end of the long development cycle.

On the other hand, agile projects start with the smallest set of most critical customer requirements to initiate a project (Nerur and Balijepally, 2007. Kakar, 2015). ASD is organized in a way that enables it to master change and uncertainty. It works on the principle of developing working products in multiple iterations. "Users review actual working product at demonstrations instead of paper reviews or reviews of prototypes done in plan-driven methods" (Nerur, Mahapatra and Mangalraj, 2005). These working products become the basis for further discussions and the team uses the latest feedback from relevant stakeholders to deliver the business solution. As the solution emerges through working products, the application design, architecture, and business priorities are continuously evaluated and refactored. A summary of ASD practices derived from Lean and Agile manufacturing is summarized (Table 3).

INSIGHTS

This comparative study finds supporting evidence in both Agile principles and practices that ASD derives its theoretical roots from agile manufacturing. The similar evolutionary paths of manufacturing and SD culminating in the agile methods are due to similar issues faced by both the disciplines. The tayloristic practices were primarily introduced to bring in efficiency and control over the production process. Lean practices were introduced to conserve resources and enhance customer value. Agile practices were introduced to manage uncertainty and change. Further, from a review of manufacturing literature the study identifies 8 facets of agility: (1) minimize waste, (2) first-time quality, (3) flexible production lines, (4) continuous improvement, 5. enriching the customer, 6. Cooperating to enhance competitiveness, 7. Organizing to master change, 8. Leveraging the impact of people and information.

Reference disciplines are usually more mature than the software engineering discipline because they have a longer history (Niederman, Gregor, Grover, Lyytinen and Saunders, 2009). They can therefore be gainfully used to understand and predict software development methods and outcomes. However, the study also calls to attention that while the application of agile principles and practices have been a welcome development, SD has implemented its own flavor of agility which contrasts with the agility principles of manufacturing.

Firstly, manufacturing agility is a philosophy and not a set of principles and practices (Gunasekharan et al., 2002). It is applicable throughout in the business-wide context and not to a specific part such as the production process (Katayama and Bennet, 1999). By implementing Agility at the project level, the software organizations may be unwittingly falling into a 'social trap' a phenomenon in which individuals or groups face the prospect of adopting seemingly beneficial behaviors that have negative consequences over time or for a larger collective (Platt, 1973, Kakar, Hale, Hale, 2012). While restricting agility to within the confines of an SD project may seem beneficial in the short run as well as optimizing at the project level, they may adversely affect accomplishment of long term project goals and result in sub-optimization at the organization level.

By contrast, in AM organizational business processes are integrated with the production process to avoid local optimizations at the expense of agility at global level. As an organization level strategy AM is designed to respond quickly to changing customer requirements through mass customization. "It demands a manufacturing system that is able to produce effectively a large variety of products and to be reconfigurable to accommodate changes in the product mix and product designs." (Gunasekaran and Yusuf, 2002). Manufacturing system re-configurability and product variety are critical aspects of agile manufacturing. ASD can learn from this. ASD project level practices should integrate with organization level processes of business strategy, core competency and supply chain management, flexible technologies and product/ project portfolio management to improve its overall efficiency and effectiveness.

"In its fully developed form, agility in manufacturing exemplifies the collaborative capability of an organization to proactively establish virtual manufacturing where a group of independent geographically distributed firms form

suitable and temporary alliances based on complementary competencies to address customer/ market needs" (Gunasekaran and Yusuf, 2002). In fact, when 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 searching for the right word to use in their manifesto, the term Agile was suggested by one of the participants who was reading the book "Agile Competitors and Virtual Organizations: Strategies for Enriching the Customer" at that time (Rigby, Sutherland and Takeuchi (2016). This bedrock strategy and core principle of agile manufacturing of implementing agility beyond to production process to the organization and beyond to derive maximum benefits is not well developed in ASD.

Secondly, manufacturing did not make a total break from the past during its evolution. For example, agile manufacturing although advocating organization level flexibility in response to uncertainty in customer/ supply chain;/ market requirements never abandoned the useful lean manufacturing and tayloristic principles and practices such as assembly line, common parts, modular design, and defined production processes. The current trend in hybridization, of integrating the practices of plan driven and ASD methods as a way forward was already well understood in AM. However, ASD at its inception and many years thereafter was presented as revolutionary with a total disregard of plan-driven practices that came before it (Boehm and Turner, 2003).

It helps resolve an apparent paradox of scripted processes and flexible and responsive development practices. Principles of lean/ agile manufacturing teach us that rather than treating them as opposite elements, it is the detailed, well-defined processes that make flexibility and creativity possible (Spear and Bowen, 1999). Successful organizations are known to successfully manage the seemingly opposing elements of innovation and efficiency, and exploration and exploitation (Katila and Ahuja, 2002; He and Wong, 2004; O'Reilly and Tushman, 2004; Gibson and Birkinshaw, 2004). Such ambidextrous organizations recognize and focus of both the organic and mechanistic structures within the organization. While the mechanistic structures help attainment of goals related to process and efficiency, the organic structures help attainment of goals related to flexibility, adaptability and innovation (Burns and Stalker, 1961; Duncan, 1976; O'Reilly and Tushman, 2004; Tushman and O'Reilly, 1996; He and Wong, 2004; Jansen, Van den Bosch and Volberda, 2005). By contrast there was almost a complete unlearning of plan-driven practices by ASD methods (Boehm and Turner, 2003). This misinterpretation of Agility in the Agile Manifesto resulted in almost a decade or more of course correction.

Agile Manufacturing	Agile Software Development
Agility in Manufacturing is a philosophy not a set of practices (Gunasekaran et al. (2002)	The concept of Agility in software development evolved from a set of practices and was driven by practitioners (Conboy, 2009)
AM is a busines wide context (Goldman and Nagel, 1993)	ASD restricted to software development projects (Conboy and Morgan, 2010)
AM is focused on design (new product development process), production, sourcing, distribution, and temporary alliances to meet customer/ supply chain/ market requirements.	ASD focused primarily on production process and activities of software development
The ideal of AM is virtual manufacturing and mass customization through modularization and late differentiation	No mention of the concept of mass customization in ASD (Ketunen, 2009) even today and virtual manufacturing is used in a very limited sense of people working together on software development projects across locations, time zone and cultures as in global software development.
AM was an evolutionary concept; did not disown the useful Lean and Mass production methods but further built on them	ASD represented a dichotomic split between agile and every other method that went before and was projected as revolutionary (Boehm and Turner, 2003)
The concept of Agility in AM has matured (Conboy, 2009)	The concept of Agility in ASD is still evolving with the research efforts current trend focused to address scalability, global agile development, distributed agile development, Agile-DevOps, Agile automation, automated testing and continuous integration (see Ebert, Gallardo Hernantes, Serrano, 2016; Alqudah and

	Razali, 2016; Dingsøyr and Lassenius, 2016)

Table 4. Differences in the concept of Agility in Manufacturing and Software Development

Thirdly, the final product which reaches the end-user in AM is the outcome of both, the product development as well as the manufacturing process. Software Development is not just a production process. It also includes the ideation, concept testing and design associated with a typical product process. There is thus a need for judiciously integrating practices from both product development and manufacturing in ASD. Consider software as a product requiring both good design and efficient production – the two activities are not mutually exclusive in but complementary. The use of product development practices could be positioned during the feasibility, concept, architectural and design phases of the project while leaving the actual development and testing to the more rigorous production practices.

Agility in manufacturing in its fullest expression deploys structured and unstructured upstream and downstream processes for product design and production. The structured processes include practices for concurrent engineering, mass customization, product portfolio management and supply chain management. The structured processes are backed by an organization level culture promoting internal and external collaboration; cross-functional communication, coordination and knowledge sharing; customer/ market focus. Further, an environment is created at team level for enhancing cohesion, reflexivity, self-organization and conflict resolution in work groups. Agile organizations recognize the value of both organic and mechanistic structures in managing uncertainty in customer demand and turbulence in the competitive landscape by providing variety in products and services aligned with the organization's strategic goals. The goal of AM is to design, manufacture, distribute, sell and service a variety of products at low cost and high quality so that customers find exactly what they want and reap the benefits of customization.

CONCLUSION

On the occasion of the 20th anniversary of the Agile Manifesto (2001) we find by hindsight that with a deeper understanding of agility in AM would have saved us years in its evolution to the present state. A correct interpretation of Agility would help the ASD realize its full potential in the future quickly without reinventing the wheel and without much experimentation. The main obstacle is that most literature on ASD due to its narrow project focus have tended to be written with the overriding assumption that the projects are managed as single projects. This does not reflect the real-life situation as project boundaries are pliable and go beyond the project to the level of the virtual enterprise to address the needs of the customer. An organization manages a basket of projects each having different priorities within budgeted resources. The projects include development projects, deployment projects and maintenance projects. New projects are continually added to the basket and existing projects prematurely discontinued or retired in alignment with the strategic goals of the organization. Just as an individual project fulfills customer requirements by building them into software to provide value to the customer, a software organization fulfills its business goals through its products and services using portfolio management to maximize the business value for the organization.

Further, as Conboy and Morgan (2010) noted a decade earlier, ASD has not focused on the role of other stakeholders besides the customer. They argued that a single customer/ user representative on the agile development team is too narrow a focus to adopt. There was also no mention of sub-contractors, suppliers, service providers and value-added resellers. This lacuna continues to persist today despite other developments in ASD. The root of this problem can be traced to the misinterpretation that has prevailed about agility in the context of software development. Lean and agile manufacturing focus on creating processes at the level of supply chain for rapid mass customization of products through modularization and late differentiation. The ultimate goal is flexibility in meeting the needs and desires of individual customers at low cost and high quality.

This ideal is stated evocatively by Toyota where Toyota visualizes its ideal plant as "one where a Toyota customer could drive up to a shipping dock, ask for a customized product or service, and get it at once at the lowest possible price and with no defects. To the extent that a Toyota plant or a Toyota worker's activity falls short of this ideal, that shortcoming is a source of creative tension for further improvement efforts" (Spear and Bowen, 1999). Agile Software development does not even talk about mass customization as a goal. Until that is done and the agile processes to accomplish that is understood, agility may not find its full expression in software develo0pment and efforts in the area of hybrid methods, global software development will either fail or produce sub-optimal results or achieve maturity through trial and error after a long time and struggle.

REFERENCES

1. Abrahamsson, P., Conboy, K. and Wang, X. (2009). 'Lots done, more to do': the current state of agile systems development research, *European Journal of Information Systems* (18), pp. 281–284.
2. Ambler, S. W. (2006). "Survey Says: AgileWorks in Practice," *Dr. Dobb's Portal: Architecture & Design*.
3. Andreeva, N. (2008) 'Lean production and agile manufacturing –New systems of doing business in the 21st century', AJII.
4. Babbage, C. (1835). *On the economy of machinery and manufacturers*. London: Charles Knight.
5. Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. First ed. Addison-Wesley Professional.
6. Bemer, R. W. (1969) "Position Papers for Panel Discussion -- The Economics of Program Production," Information Processing 68, Amersterdam, North-Holland, pp. 1626-1627.
7. Bemer, R. W. (1968) "The economics of program production", *in Proceedings of IFIP Congress (68)*, pp. 13-14.
8. Conboy, K. (2009). "Agility from first principles: reconstructing the concept of agility in information systems development," *Information Systems Research* (20:3), pp. 329–354.
9. Conboy, K. and Morgan, L. (2010). "Future research in agile systems development: applying open innovation principles within the agile organization," In *Agile Software Development* pp.223-235.
10. Cusumano, M. F. (1989) "The software factory: a historical interpretation", *IEEE Software Magazine, pp.* 23-30.
11. De Souza, S. C., Anquetil, N., and De Oliveira, K. M. (2005) A study of the documentation essential to software maintenance, Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting andDesigning for Pervasive Information (SIGDOC 2005), Coventry, UK, 68-75.
12. Dybå, T. and Dingsøyr, T. (2008). Empirical studies of agile software development: a systematic review, *Information and Software Technology* (50: 9–10), pp. 833–859.
13. Dyba, T. and Dingsoyr, T. (2009). "What do we know about agile software development?," *Software, IEEE* (26:5), pp. 6–9.
14. Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. *Ieee Software*, *33*(3), 94-100.
15. Florac, W. A. and Carleton, A. D. (1999). Measuring the Software Processes: Statistical Control for Software Process Improvement. Addison Wesley, 1999.
16. Fowler, M., and Highsmith, J. (2001). "The Agile Manifesto," *Software Development* (9), pp. 28-32.
17. Freeman, P. (1992) "Lean concepts in software engineering," *IPSS-Europe International Conference on Lean Software Development, Stuttgart*, pp. 1-8, 1992.
18. Gibson, C. B. and Birkinshaw, J. (2004). The antecedents, consequences, and mediating role of organizational ambidexterity, *Academy of Management Journal* (47:2), pp. 209–226.
19. 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.
20. 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.
21. Hannemyr, G. (1999). "Technology and Pleasure: Considering Hacking Constructive," *First Monday* (4), pp. 2.
22. He, Z. and Wong P. (2004). Exploration vs. exploitation: An empirical test of the ambidexterity hypothesis, *Organization Science* (15:4), pp. 481–494.
23. Hines, P., Holweg, M. and Rich, N. (2004). "Learning to evolve: a review of contemporary lean thinking," *International Journal of Operations & Production Management* (24:10), pp. 994-1011.
24. Hines, P., Silvi, R. and Bartolini, M. (2002), "Demand chain management: an integrative approach in automotive retailing", Journal of Operations Management 20*3), pp. 707-28.

25. Huh, W. T. (2001). "Software process improvement: operations perspectives," In *Management of Engineering and Technology, PICMET'01. Portland International Conference* (1), pp. 428-429.
26. Iacocca Institute. (1992). *21 st Century manufacturing strategy*, Lehigh University, Bethlehem, PA.
27. Jacobson, I. and Spence, I. (2009). "Why we need a theory for software engineering," Dr. Dobb's Journal.
28. Kajko-Mattsson, M., Lewis, G. A., Siracusa, D., Nelson, T., Chapin, N., Heydt, M., Nocks, J. and Snee, H (2006). "Longterm Life Cycle Impact of Agile Methodologies," In *Proceedings of the 22nd IEEE International Conference on Software Maintenance (Philadelphia, Pennsylvania, USA), IEEE Computer Society*, pp. 422-425.
29. Kakar, A.K. (2012) A theory of software development methodologies. In Proceedings of the Southern Association for Information Systems Conference, Atlanta.
30. 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.
31. Kakar. A. (2017a). Assessing Self-Organization in Agile Software Development Teams. Journal of Computer Information Systems, Vol. 57, No. 3, 208-217.
32. Kakar, A. K. (2017b). "Investigating the Prevalence and Performance Correlates of Vertical Versus Shared Leadership in Emergent Software Development Teams," Information Systems Management, 34(2), 172-184.
33. 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).
34. Kakar, A. K. (2020, A theory of effectiveness of agile software development. In *Proceedings of the American Conference of Information Systems.*
35. Katayama, H., D. Bennet. 1999. Agility, adaptability and leanness: A comparison of concepts and a study of practice. Internat. J. Production Econom. 62 43–51.
36. Katila, R. and Ahuja, G. (2002). Something old, something new: A longitudinal study of search behavior and new product introduction, *Academy of Management Journal* (45:6), pp. 1183–1194.
37. Kettunen, P. (2007). Extending software project agility with new product development enterprise agility, *Software Process: Improvement and Practice* (12:6), pp. 541-548.
38. Melnik, G. and Maurer, F. (2006). "Comparative analysis of job satisfaction in agile and non-agile software development teams," *XP2006*.
39. Mirsky, J. and Nevins, A. 1952. *The World of Eli Whitney*. The Macmillan Company, New York.
40. Moe, N. B., Aurum, A. and Dybå, T. (2012). Challenges of shared decision-making: A multiple case study of agile software development. *Information and Software Technology* (54:8), pp. 853-865.
41. Moe, N. B., Dingsøyr, T. and Dybå, T. (2009), Overcoming Barriers to Self-Management in Software Teams, IEEE Software (26:6), pp. 20–26.).
42. Nerur, S., Mahapatra, R., and Mangalaraj, G. (2005). "Challenges of migrating to agile methodologies", *Communications of the ACM*, pp. 72–78.
43. Nerur, S., and Balijepally, V. 2007. "Theoretical reflections on agile development methodologies," *Communications of the ACM* (50:3), pp. 79–83.
44. Niederman, F., Gregor, S., Grover, V., Lyytinen, K. and Saunders, C. (2009). ICIS 2008 Panel report: IS has outgrown the need for reference discipline theories, or has it? *Communications of the Association for Information Systems*, (*24:*1), pp. 37.
45. Ohno, T. (1988). The Toyota Production System: Beyond Large-Scale Production, Productivity Press, Portland, OR.
46. O'Reilly, C. A., III & Tushman, M. L. (2004). The ambidextrous organization. *Harvard BusinessReview* (82)4, 74–81.
47. Paetsch, F., Eberlein, A., and Maurer, F. (2003). "Requirements Engineering and Agile Software Development," *Proceedings of the 12th IEEE international Workshops on Enabling*.

48. Paulk, M. C., Weber, C. W., Curtis, B. and Chrissis. M. B. (1995). The Capacity Maturity Model: Guidelines for Improving the Software Process. Addison Wesley.
49. Platt, J. (1973. Social traps, *American Psychologist 2*, 641-651.
50. Poppendieck, M., and Poppendieck, T. (2003). *Lean software development: An agile toolkit*. Addison-Wesley Professional.
51. Poppendieck, M. and Cusumano, M. (2012) "Lean Software Development: A Tutorial", *IEEE Software*,.(29:5), pp.26-32.
52. Royce, W. W. (1970). "Managing the development of large software systems," In *proceedings of IEEE WESCON* (26, 8).
53. Scrum Alliance. (2008). World Wide Web electronic publication, http: //www.scrumalliance.org/view/scrum_framework.
54. Smith, A. (1776). *The wealth of nations*. Republished in 1974. Harmondsworth, UK: Penguin.
55. Spear, S. and Bowen, H. K. (1999). "Decoding the DNA of the Toyota production system." *Harvard Business Review* (77), pp. 96-108.
56. Stammel, J., Durdik, Z., Korgmann, K., Weiss, R. and Koziolek, H. (2011) Software Evolution for Industrial Automation. Systems: Literature Overview, *Karlsruher Institut für Technologie*.
57. Susman, J. I. (1976). Autonomy at work: A socio-technical analysis of participative management. New York: Praeger.
58. Taylor, F. W. (1911). *The principles of scientific management*, New York: Harper and Bros.
59. The Agile Manifesto. (2001). http://agilemanifesto.org/.
60. Turk, D., Robert, F. and Rumpe, B. (2005). "Assumptions underlying agile software-development processes," *Journal of Database Management (JDM)* (*16:*4), pp. 62-87.
61. Turk, D., France, R., & Rumpe, B. (2014). Limitations of agile software processes. *arXiv preprint arXiv:1409.6600*.
62. Womack, J.P., Jones, D.T. and Roos, D. (1991), The Machine that Changed the World, Harper Perennial, New York.

About the author(s)

Ashish Kakar: Ashish Kakar has an BS (Computer Science) from National University of Singapore and a MS (MIS) degree from Johns Hopkins University. He is now pursuing his MS (Computer Science) by Research from Texas Tech University. He has around a decade of experience in the Indian software industry. He has published papers in numerous conferences and journals such as Eservice and PAJAIS