Fall 9-11-2020

A Brief History of Software Development and Manufacturing

Ashish Kakar
Texas Tech University, ashish.kakar@ttu.edu

Akshay Kakar
University of Houston, akakar@central.uh.edu

A BRIEF HISTORY OF SOFTWARE DEVELOPMENT AND MANUFACTURING

Ashish Kakar
Texas Tech University
ashish.kakar@ttu.edu

Akshay Kakar
University of Houston
akakar@uh.edu

ABSTRACT

In this article we discover the roots and maturation of software development methods and practices through a comparative study. We notice that the evolution of software development methods has mirrored the evolution in manufacturing paradigms. Further, investigations reveal that the change software development methods have lagged the change in manufacturing paradigms indicating the source of inspiration for software development and practices is manufacturing and not the other way around. This investigation is useful and timely, especially in the context of plan-driven versus agile methods conundrum. It helps us acquire an in-depth understanding of how software development methods originated, why some of them have prevailed while others have not. Further, these insights help us assess the relevance of current practices and methods of software development and predict their future trajectory.

KEYWORDS

Software Development, Manufacturing, Agile Software Development, Plan-driven Software development

EVOLUTION OF SOFTWARE DEVELOPMENT AND MANUFACTURING PARADIGMS

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.

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. The 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 (Boehm, 1988) 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, software development turned to a more than fifty-year-old paradigm, called "Scientific Management" (Taylor, 1911). Frederick Winslow Taylor (1911) introduced Scientific Management with the aim of controlling all work activities whether simple or complicated to improve manufacturing quality and productivity. The methods suggested by Taylor focused on task simplification and time and motion studies by industrial engineers aimed at increasing specialization and standardization of work.

As applied to software development, Scientific Management, 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.

By the late 1960s, the term 'software factory' was in popular use and became associated with computer-aided tools, management-control systems, modularization, and reusability (Cusumano, 1989). 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).

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, programming standards, code inspections and productivity metrics. They encouraged division of labor leading to specialized roles of business analysts, system architects, programmers and testers.

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 (Kakar and Kakar, 2014; Kakar and Kakar, 2017f). 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 (Kakar and Kakar, 2018b; Kakar and Kakar, 2018c; Kakar, 2015; Kakar and Kakar, 2017f, Kakar and Kakar, 2018d). 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 (Kakar, 2017d; Kakar and Kakar, 2017f).

Lean Manufacturing and Lean Software Development

Lean manufacturing originated on the shop-floors 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. 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 focuses 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. Lean principles were first applied to software development in the 90s (Freeman, 1992), well before the Agile principles (see Table 3). 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.

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 similar to 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.

Agile Manufacturing and Agile Software Development

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. 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 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 have favored agile solutions (Goldman et al., 1995).

Agile manufacturing approaches focus on addressing 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.

In line with these developments in manufacturing, ASD (Agile Software development) began as a countermovement to the Taylorist software development processes like the Waterfall Model or the V-Model. 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 (Kakar, 2014). As a result. the validation of the product happens only at the requirement gathering stage and at the end of the long development cycle.

"On the other hand, agile projects work on minimum critical specification." (Nerur, Mahapatra and Mangalraj, 2005) Agile projects start with the smallest critical set of requirements to initiate the project. They work on the principle of developing working products in multiple iterations. "Users review actual working product at demonstrations instead of paper reviews or review 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 (Kakar, 2018a; Kakar and Carver, 2012).

The evolution of software development approaches and the corresponding manufacturing paradigms are summarized in 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 SDMs

The history of software development and manufacturing thus indicate that software development principles and practices were adapted from manufacturing. We further assess the validity of this premise by comparing the practices of two major software development methods - plan-driven (or Tayloristic) and agile methods - with the corresponding practices in the two major manufacturing paradigms. Table 2 summarizes the findings for Tayloristic

methods of software development and manufacturing, and Table 3 summarizes the findings for agile methods of software development and manufacturing.

	Tayloristic Practices of Mass Production	**Adaptations by Plan-driven methods of Software Development**
	Goals and Means	
1	Efficiency of Production Process	Focus on Cost, Quality, Schedule
2	Focus on Time and Motion Study, Task Simplification	Adherence to Productivity Norms e.g. Lines of Code (LOC)/ Programmer week, Preventing schedule slippage, Low Defect/
3	Specialization through Division of Labor	Specialized Roles: Designers, Programmers, Testers
4	Standardization, Common Parts	Coding Standards, Code Reuse
	Process Strategy	
1	Defined Production Process	Upfront Planning, Defined Process
2	Long Production Runs	Long gap between requirements capture and delivery of IS product
3	Management control of production process	Reviews, Audits and Inspection
4	Push approach, Assembly line	Sequential phases, Waterfall approach
5	Automation of Production Process e.g. CAM (Computer Aided Manufacturing)	Automation of development processes: e.g. CASE (Computer Aided Software Engineering) Tools
	Quality Control	
1	Objective: Meeting Product Specifications	System Requirement Specification and Verification
2	Means: Defect Detection and Correction, Statistical Control	End of Line Testing. Statistical Process Control

Table 2. Tayloristic practices adopted by Software Development

ASD practices from Lean Manufacturing	**ASD practices from Agile Manufacturing**
Minimizing waste (adapted 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
Use of User Stories – feature descriptions written from the customer perspective.	Use of user stories – feature descriptions written from the customer perspective
Extra Processing Steps: Code directly from 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 (also see Kakar, 2017a)
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 than bureaucratic control (also see Kakar, 2017b)
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 (also see Kakar, 2017c)
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

DISCUSSION AND CONTRIBUTION

As Jacobson and Spence (2009) point out, theoretical roots help us glean the methodology-independent "truths" of software development. This comparative study, a first of its kind, traces the origins and maturation of the concepts and practices in the disciplines of manufacturing and software development and finds noteworthy similarities between them (Table 3). The study provides evidence that the evolution of software development has trailed the developments in manufacturing. From Table 2, we can see that although evolution in software development methods lags evolution in manufacturing paradigms, it eventually catches up. Further, the lag time is progressively reducing. This discovery has useful implications for research and practice. Three benefits are readily identifiable.

Firstly, the increasing popularity of Agile Software Development (ASD) compared to plan-driven methods have perplexed many. Many still believe that ASD methods are a passing fad. However, the findings of this study show that ASD can be viewed as a natural progression of evolution in software development methods (see table 2). Additionally, this study also shows that ASD is currently at the apex of evolution in software development methods and is likely to increasingly displace plan-driven methods of software development as lean/ agile manufacturing paradigms have displaced Tayloristic principles of mass production over the last century.

Secondly, the findings of the comparative study provide a glimpse of the future trajectory in evolution of software development methods. ASD methods are not without limitations. Agile methods continue to be seen as restricted to small, co-located development teams and not well-suited for developing large enterprise software. The answer to addressing these limitations may lie in tracking the relevant practices of manufacturing from where many of the current software development practices in general and ASD practices in particular are drawn.

For example, in its fully developed form, agility in manufacturing exemplifies the collaborative capability of an organization to proactively establish virtual manufacturing (Gunasekaran and Yusuf, 2002) where a group of independent geographically distributed firms form suitable and temporary alliances based on complementary competencies to address customer/ market needs. Currently the agile principles and practices do not even provide the basic guidelines and processes for sub-contracting or outsourcing, leave alone address the sophisticated processes of a full-blown virtual manufacturing. But aligned with the developments in manufacturing, ASD in future

will likely incorporate the advanced collaborative practices of virtual manufacturing as it evolves to address the challenge.

Thirdly, a need continues to be expressed amongst both practitioners and researchers for a coherent understanding of what constitutes agility in software development. By tracing the roots of the concept of agility in manufacturing and discovering their implementation in agile principles and practices this study identifies eight facets of agility - 1. Organizing to master change; 2. Enriching the customer; 3. Cooperating to enhance competitiveness; 4. Leveraging the impact of people and information; 5. Minimize waste; 6. Perfect first-time quality; 7. Flexible production lines; and 8. Continuous improvement. Future research may develop individual scales and/ or a composite agility index based on these 8 facets of agility to measure the agility of ASD projects to pinpoint areas of improvement.

REFERENCES

1. Baker, J. (1996). "Agility and Flexibility, What 'the Difference?," *Working Paper* (5:96), Cranfield University.
2. Bemer, R. W. (1969) "Position Papers for Panel Discussion -- The Economics of Program Production," Information Processing 68, Amersterdam, North-Holland, pp. 1626-1627.
3. Boehm, B. W. (1988). "A spiral model of software development and enhancement," *Computer*,(*2:* 5), pp. 61-72.
4. Cusumano, M. A. (1988). *The software factory: an interpretation* (3). Afred P. Sloan School of Management, Massachusetts Institute of Technology.
5. 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.
6. 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.
7. Hannemyr, G. (1999). "Technology and Pleasure: Considering Hacking Constructive," *First Monday* (4), pp. 2.
8. 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.
9. Huh, W. T. (2001). "Software process improvement: operations perspectives," In *Management of Engineering and Technology, PICMET'01. Portland International Conference* (1), pp. 428-429.
10. Iacocca Institute. (1992). *21 st Century manufacturing strategy*, Lehigh University, Bethlehem, PA.
11. Jacobson, I. and Spence, I. (2009). "Why we need a theory for software engineering," Dr. Dobb's Journal.
12. Kakar, A. K. S., & Carver, J. (2012). Best practices for managing the fuzzy front-end of software development (sd): Insights from a systematic review of new product development (NPD) literature. In *Proceedings of International Research Workshop on IT Project Management* (p. 14).
13. Kakar, A. K. (2014). When form and function combine: Hedonizing business information systems for enhanced ease of use. In *2014 47th Hawaii International Conference on System Sciences* (pp. 432-441). IEEE.
14. Kakar, A. K. (2015). Investigating the penalty reward calculus of software users and its impact on requirements prioritization. *Information and Software Technology*, *65*, 56-68.
15. Kakar, A. K. (2017a). Do reflexive software development teams perform better?. *Business & information systems engineering*, *59*(5), 347-359.
16. 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.
17. Kakar, A. K. (2017c). Assessing self-organization in agile software development teams. *Journal of computer information systems*, *57*(3), 208-217.
18. Kakar, A. K. (2017d). How do perceived enjoyment and perceived usefulness of a software product interact over time to impact technology acceptance?. *Interacting with Computers*, *29*(4), 467-480.
19. Kakar, A. K., & Kakar, A. (2017e). A General Theory of Technology Adoption: Decoding TAM from a User Value Perspective. *Southern Association of Information Systems*.
20. Kakar, A. K., & Kakar, A. (2017f). COSTS LOOM LARGER THAN GAINS: AN INVESTIGATION OF CONSUMERS'ONLINE VERSUS INSTORE SHOPPING PREFERENCES. *AMA Winter*.

21. Kakar, A. K. S. (2018a). Engendering cohesive software development teams: Should we focus on interdependence or autonomy?. *International Journal of Human-Computer Studies*, *111*, 1-11.
22. Kakar, A., & Kakar, A. K. (2018b). Assessing Shopper's Penalty Reward Calculus in Online versus Instore Shopping. *e-Service Journal*, *10*(3), 24-45.
23. Kakar, A., & kumar Kakar, A. (2018c). Is team cohesion a double edged sword for promoting innovation in software development projects?. *Pacific Asia Journal of the Association for Information Systems*, *10*(4).
24. Kakar, A. K., & Kakar, A. (2018d). IS THE TIME RIPE FOR BRANDING OF SOFTWARE PRODUCTS.
25. Mellroy, M . D. (1968) "Mass produced software components", *Report NATO Conference. on Software Engineering, Garmisch,* pp. 138-152.
26. Nerur, S., Mahapatra, R., and Mangalaraj, G. (2005). "Challenges of migrating to agile methodologies," *Communications of the ACM*, pp. 72–78.
27. Poppendieck, M., and Poppendieck, T. (2003). *Lean software development: An agile toolkit*. Addison-Wesley Professional.
28. Royce, W. W. (1970). "Managing the development of large software systems," In *proceedings of IEEE WESCON* (26, 8).
29. Taylor, F. W. (1911). *The principles of scientific management*, New York: Harper and Bros.
30. Womack, J., Jones, D. and Roos, D. (1990). *The Machine that Changed the World*, Rawson Associates, New York, NY.