2012

Teaching Theories Underlying Agile Systems Development

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

Joanne Hale
University of Alabama, jhale@cba.ua.edu

David Hale
University of Alabama, dhale@cba.ua.edu

TEACHING THEORIES UNDERLYING AGILE SYSTEMS DEVELOPMENT

Adarsh Kumar Kakar
Computer Information Systems Department
Alabama State University
akakar@alasu.edu

Joanne Hale
Management Information Systems Department
University of Alabama
jhale@cba.ua.edu

David Hale
Management Information Systems Department
University of Alabama
dhale@cba.ua.edu

Abstract:

Presently Agile methods courses taught in universities focus primarily on providing hands-on experience of the process of development but ignore the evolution of, and theories behind, the Agile practices. “Without theory we are just groping in chaos” (Deming, 1986). Knowing the ‘why” in addition to the “how” of Agile methods will help develop reflective skills and give students an edge as they transition to the rapidly evolving real world of IS. In this article a set of relevant theories that can be included as a module in an Agile method course is outlined. An exposure to theories underlying Agile methods help students appreciate the relevance of the principles and practices of the Agile approach and develop authentic problem solving skills.

Keywords: Agile Methods, Theory and Paradigms, Course Module

I. INTRODUCTION

Both theoretical and experiential knowledge are necessary preconditions for successful performance of one’s job. Theory and practice support each other in a way that includes an understanding of the rationale according to which tasks should be carried out and an understanding of the boundary conditions of the given job. Thus, knowledge that is useful at work includes the dimensions of both practical knowledge and theoretical understanding. Recent accounts on the development of expertise have emphasized that a combination of these dimensions is of fundamental importance (e.g. Leinhardt *et al.,* 1995).

This is because work-based learning is not a unified phenomenon but varies in different contexts and between actors. Unless the students develop reflective skills they will not be able to apply the knowledge and skills developed in the classroom to the real world. Only providing hands-on experiential knowledge of the process of development is not enough to develop the self-regulative knowledge including metacognitive and reflective skills. Formal or theoretical knowledge is also essential.

But this is easier said than done. In the absence of academic work in the area the domain of Agile methods has remained largely atheoretical. There are no theories or models to provide guidance only a set of principles and practices. As a result Agile method courses focus only on the narrow process/ practice perspective and fail to prepare students with the depth needed to solve real world problems.

To fill the gap, this article traces the evolution of Agile methods and identifies the relevant theories that should be taught as a module in an Agile methods course. In the authors’ own experience, this provides student with a structural framework that allows them to make sense of their hands-on experience. Theories develop reflective thinking and serves as a benchmark against which the learnings from an agile development method course can be measured against to determine what they are doing right and where

they are going wrong. Students are then better equipped to apply learning at the work-place under various circumstances in the real world.

II. APPROACH

Agile methods represent a major departure from traditional, plan-based approaches to software engineering (Dyba and Dingsoyr, 2009). The issue of how software development should be organized in order to deliver faster, better, and cheaper solutions has been discussed in software engineering circles for decades. Many remedies for improvement have been suggested, from the standardization and measurement of the software process to a multitude of concrete tools, techniques, and practices. Recently, many of the suggestions for improvement have come from experienced practitioners, who have labeled their methods agile software development. This movement has had a huge impact on how software is developed worldwide (Dyba and Dingsoyr,, 2009). The increasing popularity of agile methods makes it imperative that agile software development should be taught at university level (Hazzan and Dubinsky, 2007).

Although not explicitly identified or stated, Agile methods are an amalgamation of many theories and concepts. The module introducing theories underlying the agile methods could begin with comparison of how the evolution of Systems Development Methods (SDMs) mirror those of other manufacturing paradigms. This will provide a context and a fresh perspective to students that will affirm their knowledge of SDMs and Agile methods by comparison. Students will learn to appreciate how a popular theory or system is wholly upended (Kuhn 1962) giving rise to new theories and paradigms. Additionally they will learn to appreciate how concepts emerge and will help them to actively (re)construct concepts on their own.

Building on this foundation the instructor can then explore the theories underlying the Agile manifesto and its 12 principles. The relevant concepts and theories that in the author’s experience are useful to the students are the Job design theory, the marketing concept, socio-technical system perspective, process control theory, the theory of emergence and approach-avoidance theory. The descriptions of the evolution of SDMs and the theories underling Agile methods provided in this article are illustrative and not exhaustive and are meant to provide the reader with an idea of what the suggested theory module of an Agile methods course should contain.

III. EVOLUTION OF AGILE METHODS

Craftsmanship and Code-and-fix

In the 1950s, people working with computers had much in common with artists, artisans and craftsmen before the industrial revolution (Hannemyr,1999). There was room for creativity and independence. Management methods of control were not yet developed. There was no clear division of labor. Skilled programmers, like all good craftsmen, had intimate knowledge and understanding of the systems they worked with. Programmers were able to get by with this type of development for two reasons. First, no better way had been developed, and second, software was not that complex. This did not last. As software grew more complicated and organizations relied on computers for more of their operations, including finances and even human lives, this laissez faire approach to programming gave way to more disciplined methods. By the mid-sixties, management wanted to bring computer work in line with other industrial activities, which essentially meant that they wanted programming to be part of a managed and controlled process.

Taylorism and Waterfall

To accomplish this, software developers turned to a more than fifty year old paradigm, called "Scientific Management" (Taylor, 1911). Scientific Management was invented by the engineer Frederick Winslow Taylor, and aimed at taking away from workers the control of the actual mode of execution of every work activity, from the simplest to the most complicated. Taylor's argument was that only by doing this could

management have the desired control over productivity and quality.

The methods advocated by Taylor were to increase standardization and specialization of work. In the computer field, this implied, among other things, the introduction of programming standards, code reviews, structured walkthroughs and miscellaneous programming productivity metrics. The taylorist methods such as the waterfall model and its variants promote strong conformance to a plan through upfront requirements gathering and upfront systems design. They also encourage strict Tayloristic division of labor and the use of role based teams of business analysts, system architects, programmers and testers (Maurer and Melink, 2005).

Although this model of development was a substantial improvement over the former model of "code-andfix" methods, the software developed was both late and did not fundamentally address the customer's real needs. Under conditions of rapidly evolving customer needs the approach of full requirements definition, followed by a long gap before those requirements are delivered is no longer appropriate. With problem complexity, changing scope and requirements, and technologies evolving during the project, developers, over time, came to understand that the "dreaded system integration phase" would not go as well as planned.

Agile and Lean Production

Agile software development began, in the early 1990s, as countermovement to the Taylorist software development processes like the Waterfall Model or the V-Model. Taylorist approaches are based on the principle that the first step in a product/ system solution is to comprehensively capture the 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 work on minimum critical specification (Nerur and Balijepally, 2007). Agile projects start with the smallest 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 in plan-driven methods. These working products become the basis for further discussions and the team works towards delivering the business solution using the latest input from customers, users, and other stakeholders. As the solution emerges through working products, the application design, architecture, and business priorities are continuously evaluated and refactored. The Agile methods such as Extreme programming, Scrum, Crystal methodologies, Dynamic Software development method (DSDM), Feature Driven Development (FDD) and Lean Software Development Method (LSDM) are based on the Agile principles (Table 1). Many of the Agile software development principles introduce in 2000s have their origins in the Lean and Agile manufacturing paradigms introduced in the 1980s and 1990s respectively.

The origins of lean thinking can be found on the shop-floors of Japanese manufacturers and, in particular, innovations at Toyota Motor Corporation (Shingo, 1981, 1988; Monden, 1983; Ohno, 1988). These innovations, resulting from a scarcity of resources and intense domestic competition in the Japanese market for automobiles, included the just-in-time (JIT) production system, the kanban method of pull production, respect for employees and high levels of employee problem-solving/automated mistake proofing. This lean operations management design approach focused on the elimination of waste and excess from the tactical product flows at Toyota (the Toyota "seven wastes") and represented an alternative model to that of capital-intense mass production (with its large batch sizes, dedicated assets and "hidden wastes").

Agile manufacturing is seen as the next step or extension in the 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 (Iacocca Institute, 1991). The origins of the "agility movement" stems from US government concerns that domestic defence manufacturing capability would

be diminished following the end of cold war in 1989. While the proposed definition of leanness is “the maximisation of simplicity, quality and economy” agile manufacturing added flexibility and responsiveness to the definition. It seeks to achieve competitiveness through rapid response and mass customization. Whereas lean methods offer consumers good quality products at low price by removing inventory and waste from manufacturing agile manufacturing is a strategy for entering niche markets rapidly and being able to cater for specific needs of ever more demanding customers on an individual basis.

IV. THEORIES

Process Control Theory

Industrial process control theory specifies two types of process control systems: defined processes and empirical processes (Schwaber, 1995). Defined processes are those which, given a certain set of inputs, and applying a certain set of controls, always attain a specified outcome and are repeatable. They are referred to as whitebox systems, as the processes are well defined and understood.

Empirical processes, on the other hand, are referred to as black-box systems. These processes are generally complex in nature, not well understood and have no defined set of controls that can be applied to repeatedly generate the desired outcome. Such processes have unpredictable outcomes, and can only achieve desired outcomes empirically. In other words, one applies a certain degree of control, measures the output, adjusts the controls and repeatedly do this until the desired outcome is finally reached - like a missile homing in on a target.

Software is considered to be such a complex system, as there is no way in which one set of controls can be put in place in order to provide a predictable or repeatable desired outcome. Some of the reasons for this lack of predictability are in large part due to the high degree of uncertainty surrounding the technology, and business requirements. Even with highly detailed upfront user interface designs, specifications and plans, the software produced often turns out different from its original intent. This can be attributed to the fact that once end users see the software and use it, they realizes there are often different and improved ways of doing things.

Therefore applying defined process methodologies to intrinsically unpredictable and unrepeatable systems does not always work. Waterfall methodologies, which most software teams are currently using, are a form of defined process, as all the unknowns are expected to be solved up-front. Waterfall methodologies assume that software development is a defined process, i.e. a well understood process. However this is often not true. Consequently, in Agile approaches, any detailed up-front effort to fully understand the problem domain is considered wasteful. If one borrows from Lean thinking, excessive upfront planning can be thought of as inventory on the shop floor, which is a liability rather than an asset. Uncertainty is not something that you can just plan away with up-front research and design. Processes have to be more fluid to deal with them.

Emergence

When faced with changing requirements and technologies, agile methodologists do not believe that a software application can be fully specified up-front. Instead, the true requirements lead towards the development of a system a customer *actually* wants (as opposed to what they initially thought they wanted) should be allowed to emerge over time. This is the rationale behind agile methodologies welcoming changing requirements, even late in the development cycle. The short iterations and customer inspection of working software from each iteration provide the mechanism to allow requirements to emerge. The goal of this flexibility is to deliver what the customer really wants even in the face of constant change and turbulence.

Job Design Theory

A methodology is a systematic way of performing a task or doing work. Therefore it is logical to look at SDMs from the perspective of job design. It opens up avenues to vast existing literature on job design and makes them available to the newer discipline of software engineering.

The literature on job design contrasts "Taylorist" jobs to the "Enriched" jobs. Fredrick Taylor (1947) viewed job design as a scientific optimization problem, where industrial engineers study the production process and devise the most efficient way to break that process into individual, precisely defined tasks. Typically, a Taylorist job is highly specialized, and workers are not encouraged to experiment, innovate, or otherwise vary the way that tasks are performed.

In the 1970's, academics such as Richard Hackman, Edward Lawler and Greg Oldham started to argue that Taylorist job design is sub-optimal (Hackman and Qldham, 1976; Hackman and Lawler, 1971; Lawler, 1973; Porter, Lawler and Hackman, 1975). Enriched jobs, by encouraging workers to learn and innovate at work, increase the motivating potential of work. Motivated workers perform tasks more accurately and are more likely to find productivity innovations that engineers overlook. In the 1980's, firms put the theory into practice by redesigning jobs, adopting self-managed teams and work groups, and creating employee participation programs like quality circles.

The Hackman and Oldham (1976) Job Characteristics Model (Figure 1) is one of the most elaborate and widely accepted theories of job design (Kiggundu, 1981), and can be used to explain the emerging trends and empirical observations in the domain of SDMs.

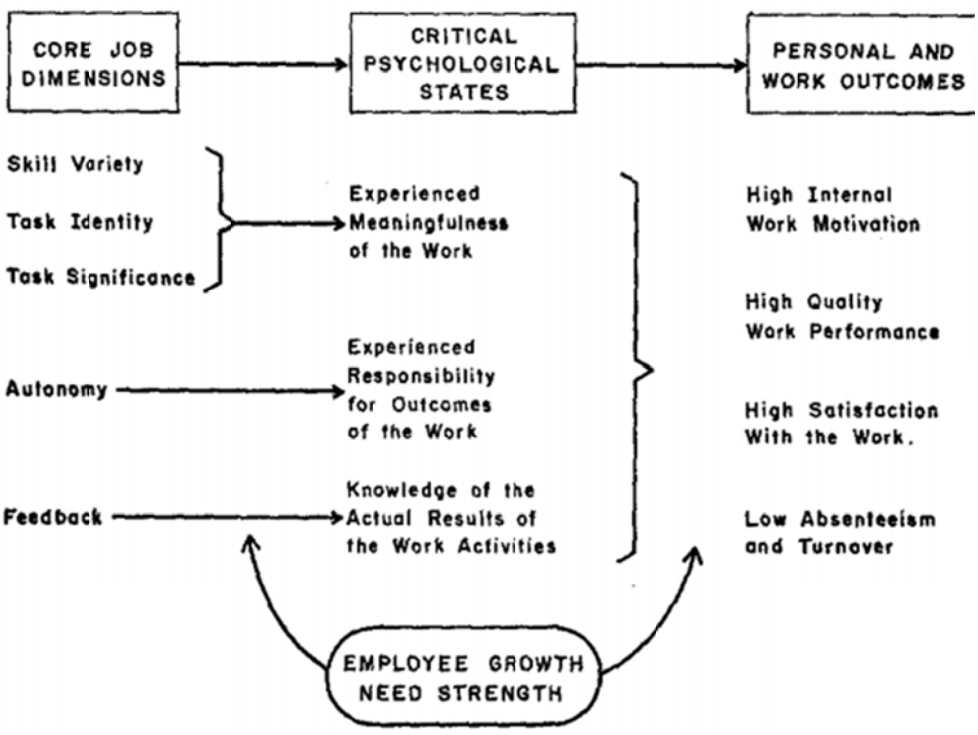

Figure 1: The Job Control Theory (JCT)

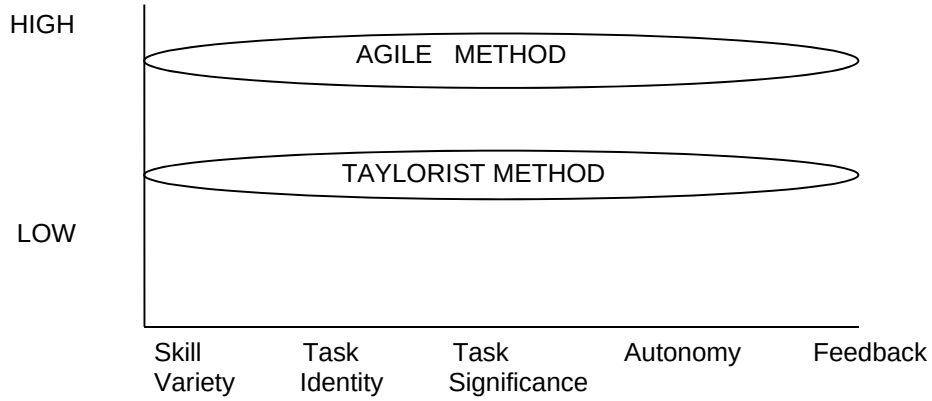

Figure 2. A comparison of Agile and Taylorist methods

Taylorist software development methods deploy specialized role based teams, with individual team members requiring less skill variety to accomplish jobs. Detailed planning is done of entire software development lifecycle activities including requirements gathering, design, construction, testing and project coordination and management activities and specialized people handle each of these tasks. The allocation of work specifies "not only what is to be done but how it is to be done and the time allowed for doing it" (Chau, Maurer, and Melnik, 2003). This reduces the autonomy of employees and shifts the focus from individuals and their creative abilities to the processes themselves.

On the other hand agile methods emphasize and value individuals and interactions over processes. Agile methods are people-centric, recognizing the value competent people and their relationships bring to software development (Nerur, Mahapatra, and Mangalaraj, 2005). People issues are at the heart of the agile movement (Boehm and Turner, 2005). The agile team works by placing people physically closer, replacing documents with talking in person and at whiteboards, improving the team's amicability and its sense of community (Cockburn and Highsmith, 2001). Tasks are not specialized to the degree of plan-driven methods. All team members are involved in coding, designing and testing thus increasing the skill variety needed to complete a task.

Agile methods move away from a deterministic/ mechanistic view of problem solving to a dynamic process characterized by iterative cycles and the active involvement of all stakeholders. Unlike the Taylorist methods, where the cycle time between requirements gathering and product release is typically very long, the gaps between customer requirements and implementation into the product in agile projects are narrowed in rapid cycles. The focus on developing working products rather than paper artifacts and components enhances task identity and task significance. Big upfront design plans and extensive documentation are of little value to practitioners of agile methods (Nerur and Balijepally, 2007). Important features of this approach include evolutionary delivery through short iterative cycles – of planning, action, reflection – intense collaboration, self-organizing teams, and a high degree of developer discretion, providing the team members autonomy as well as quick feedback on the work accomplished. The agile paradigm empowers individuals through a focus on developing working products, ownership and shorter feedback cycles (Boehm and Turner, 2005), satisfying the three psychological states of the job characteristics model, the need for meaningful work, the need to be responsible for work outcomes, and the need for performance feedback. This increases the motivating potential of work, as measured by the Motivating Potential Score (MPS), calculated by using the formula (Hackman and Oldham, 1976):

$$\text{MPS} = \frac{\text{Skill Variety} + \text{Task Identity} + \text{Task Significance}}{3} \times \text{Autonomy} \times \text{Job Feedback}$$

resulting in higher team member morale, satisfaction and productivity.

JCM is able to explain the relevance of various best practices. For example the benefits of Paired programming can be explained by the rapid feedback it provides to the developers. The benefits of developing working products in rapid iterative cycles is due to the enhanced significance of the task completed and getting early feedback from the users of the product. In addition developing whole, meaningful and working products makes it easy for developers to identify with the tasks that are fulfilled. This in turn increases the motivating potential of team members and the resulting work outcomes. This is in contrast to the work where developers are given a specification for parts of the solutions, and do not have full picture of the product this is being developed.

Approach-Avoidance Theory

Motivation to continue or pull out from a project can be viewed as an approach avoidance conflict. In approach avoidance theory, when driving forces that encourage persistence outweigh restraining forces that encourage abandonment (Brockner and Rubin, 1985) people will be motivated to continue and complete the project. When restraining forces prevail over the driving forces people will withdraw. These competing forces create a conflict over whether to continue or withdraw (Mann, 1966) impacting motivation of individuals and teams.

Derived from approach avoidance theory, the completion effect, a driving force, reflects the notion that the "motivation to achieve a goal increases as an individual gets closer to that goal" (Conlon and Garland, 1993). The completion effect is consistent with psychological research suggesting that the desire to achieve task closure, or completion, can have a significant influence on behavior (Katz and Kahn, 1966). Results from a series of experiments provide support for the completion effect (Conlon, and Garland, 1993; Garland and Conlon, 1998). Specific evidence that the motive to complete a task gets stronger as one gets closer to completion can be found in work by Lewin (1935) whose hypothesis was supported in later empirical work (Krech, 1935; Krech, Crutchfield and Liuson, 1969; Miller, 1944; Brown, 1948). These studies demonstrated that motivational intensity increased as the subjects moved closer to a desirable goal object.

If individuals are motivated to complete what they start and if this motive gets stronger as one gets closer to completion, then project completion may be a driving force behind individuals' continuing to invest efforts in projects that are already well under way. It overcomes the costs of persistence, resulting in motivated individuals and teams working towards task closure, resulting in greater probability of successful project outcomes and user satisfaction.

When the goal seems distant, and there is little visibility into project progress, uncertainty about project outcomes builds up. The restraining force of cost of persistence then dominates resulting in demotivated individuals and teams, project delays and user dissatisfaction. In a classic case, The London Stock Exchange scrapped an electronic share-transfer system known as TAURUS, despite work spanning a decade and investments estimated to be as high as 400 million pounds (Drummond, 1996; Duffy, 1993). TAURUS was originally scheduled to be operational in 1989, but was abandoned in 1993, when it became clear that the system was still several years from completion. A major factor in the decision to abandon the project was the state of project incompletion (Drummond, 1996).

Agile projects work on minimum critical specification (Nerur and Balijepally, 2007). Agile projects start with the smallest 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 in plan-driven methods. These working products become the basis for further discussions and the team works towards delivering the business solution using the latest input from customers, users, and other stakeholders. As the solution emerges through working products, the application design, architecture, and business priorities are continuously evaluated and refactored.

Iteration by iteration, everyone involved can see whether or not they will get what they want. As a result project progress is visible and the ability to decide what is to be done next is more complete, thus reducing uncertainty and giving stakeholders more confidence in the state of completion of the project. As the project moves progressively towards completion, the motivation of team members, and users who form part of an extended team in Agile projects, keeps increasing. The team members and users rapidly hit their stride with increasing motivation, impelling them to continue to invest their efforts and accelerating them towards successful project completion.

Marketing Concept

Agile development approach brings the marketing concept into software engineering by emphasizing the primacy of addressing evolving customer requirements. Keith's (1990) article, on the marketing concept is one of the earliest and most popular. It is a descriptive article illustrating the adoption of the marketing concept in an applied setting. The intuitive appeal of the concept and its successful application in practice played an important role in its acceptance. In the article, Keith describes the Pillsbury Company's evolution through three managerial phases, finally reaching what he calls a marketing control phase. His description suggests that movement from the production through the sales and later through the marketing phase has been an evolutionary process which left the organization a stronger entity. The implication for the business is that this evolutionary process is the correct one for all organizations. Customer focus is a core element of the marketing concept (Rosen, Schroeder and Purinton,1998). Theodore Levittt's (1983) seminal statement of the marketing concept argued that customer needs must be the central focus of the firm's definition of its business purpose.

Although the plan-driven approaches such as the waterfall model do emphasize that user requirements should be gathered before the design and development stage, it is not well suited to accommodate requirement changes during its development cycle which may sometimes take a few years. This approach is appropriate when requirements are stable. However in today's faced paced business context, the needs of customers evolve continuously in response to changes in environment in which they operate. Software developers with customer focus aim *to provide competitive advantage to their customers* (one of the Agile principles) by acquiring the ability to address these customer demands rapidly by developing working products in quick iterations and with minimal waste. Therefore the agile methods have more comprehensively embraced the market concept which is as relevant in today's business as it was in the 1990s.

Socio-Technical Systems Perspective

From a socio-technical systems perspective, research on self-organizing teams dates back to the Tavistock group's study of English coal miners as autonomous groups in the 1950s (Trist, 1981). Autonomous groups were described as learning systems that expand their decision space in response to every day learning. The success of these autonomous groups was largely attributed to the supporting organizational environment, an informal structure with a decentralized, participative, and democratic system of control, called concertive control (Lewin, 1948). Concertive control was argued to be an alternative to the bureaucratic control marked by an hierarchical system with rational-legal rules rewarding compliance (Baker, 1993). Self-managing teams were proposed as an exemplar of concertive control and were suggested to increase the organization's ability to respond to changing business conditions (Lewin, 1948).

Self-managing teams were described as teams made up of 10 to 15 people taking on the responsibilities of their former supervisors; whose every day activities were guided by the senior management's corporate vision; who were cross-trained individuals setting their own work schedules; who displayed increased commitment to the company; and who co-ordinated with other areas of the company (Lewin, 1948). Selfmanaging teams in a concertive organization were said to be motivated by peer-pressure as opposed to legal rules in a bureaucratic organization. The distinct synergy between the description of these self- managing teams and the theoretical concept of a self-organizing team proposed in Agile software development is inescapable (Highsmith, 2004).

Agile methodologies give the entire (extended) development team the autonomy to self organize in order to determine the best way to get the job done. Team members are not constrained by predetermined roles or required to execute obsolete task plans. Managers of agile teams place a great deal of trust and confidence in the entire team. In self organization, the emphasis is on face-to-face conversations, rather than on communicating through formal (or informal) documents. Software developers talk with software developers, business people talk with software developers, customers talk directly with either business people or software developers. Agile methodologies also advocate the use of post-mortem meetings in which team member reflect on how to become more effective. The team then tunes and adjusts its behavior accordingly.

The theories and paradigms that form the basis of agile principles can be summarized in Table 1 below:

Table 1. Agile principles and Theories underlying Agile principles

Agile Principles	Theories and Paradigms
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	Agile Manufacturing (Responsiveness), JCT (Task Significance – valuable software), Marketing concept (Customer satisfaction)
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	Agile manufacturing (Flexibility – welcome change), Marketing concept (Respond to changing customer requirements)
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	Agile manufacturing (Speed), JCT (User Feedback), Working Software (Completion Effect)
Business people and developers must work together daily throughout the project.	Marketing concept (Understand customer needs)
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	JCT (Autonomy), Socio-Technical perspective (Autonomy)
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	
Working software is the primary measure of progress.	Lean Manufacturing (Make only what is pulled by the customer, JCT (Task identity), Working software (Completion Effect)
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	
Continuous attention to technical excellence and good design enhances agility.	Quality (Lean manufacturing)
Simplicity--the art of maximizing the amount of work not done--is essential.	Simplicity (Lean Manufacturing)
The best architectures, requirements, and designs emerge from self-organizing teams.	JCT (Autonomy), Socio-Technical perspective (Self-Organizing teams)
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	JCT (Job feedback)

V. CONTRIBUTION AND CONCLUSION

It is a well-known fact that the process of software development is a complicated task, composed of many aspects, such as cognitive, social, and technical ones (Hamlet and Maybee, 2001; Tomayko and Hazzan, 2004). Accordingly, the academia has a significant and non-trivial role in the education of future software developers toward this multifaceted challenge. This work makes a call for augmenting the theoretical knowledge of SDMs. In the author's experience with teaching agile software development methods, it makes a big difference in the depth of understanding that students acquire when the theoretical aspects of agile are emphasized than when they are not. With theoretical insight students begin to understand why agile practices work and under what context. Accordingly an outline of the theoretical component is suggested which should become an integral part of any agile system development course.

VI. REFERENCES

Baker, J. (1993) "Tightening the iron cage: Concertive control in self-managing teams", *Administrative Science Quarterly*, (3)38, pp. 408-437.

Boehm, B. and R. Turner (2005) "Management challenges to implementing agile processes in traditional development organizations", *IEEE Software*, (5) 22, pp. 30–39.

Brockner, J. (1992) "The Escalation of Commitment to a Failing Course of Action: Toward Theoretical Progress", *Academy of Management Review, (1)*17, pp. 39-61.

Brockner, J. and J. Z. Rubin (1985) "The social psychology of entrapment in escalating conflicts", New York: Springer-Verlag.

Brown, J. S. (1948) "Gradients of approach and avoidance responses and their relation to motivation", *Journal of Comparative and Physiological Psychology*, 41, pp. 450-465.

Bunsen, C., Feldmann, R. L. and J. Dorr (2004) "Agile methods in software engineering education", In *5th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP2004* (Vol. 3092 Lecture Notes in Computer Science), pp. 284-293,

Chau, T., Maurer, F. and G. Melnik (2003) "Knowledge Sharing: Agile Methods vs. Tayloristic Methods", IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03), Linz, Austria, pp. 302-308.

Cockburn, A. and J. Highsmith (2001) "Agile Software Development: The People Factor", *Computer*, pp. 131-133.

Conlon, D. E., Garland, H. (1993) "The role of project completion information in resource allocation decisions", *Academy of Management Journal*, 36, pp. 402–413.

Conboy, K. and B. Fitzgerald (2004) "Toward a conceptual framework of agile methods: a study of agility in different disciplines", in: Proceedings of XP/Agile Universe, Springer Verlag.

Drummond, H. (1996) "Escalation in decision making: The tragedy of TAURUS.", Oxford, UK: Oxford University Press.

Duffy, M. (1993) "London's embarrassing mistake", *Wall Street and Technology*, 10, pp. 38-43.

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

Dybå, T. and T. Dingsøyr (2009) "What Do We Know About Agile Software Development", *IEEE Software*.

Garland, H. and D. E. Conlon (1998) "Too close to quit: The role of project completion in maintaining commitment". *Journal of Applied Social Psychology*, 28, pp. 2025–2048.

Hackman, J.R. and E. E. Lawler (1971) "Employee reactions to job characteristics", *Journal of Applied Psychology Monograph*, 55, pp. 259-286.

Hackman J.R. and G. R. Oldham (1974) "The Job Diagnostic Survey: An instrument for the diagnosis of jobs and the evaluation of job redesign projects", *JSAS Catalog of Selected Documents in Psychology (Ms. No. 810)*, 4, pp. 148.

Hackman, J.R, and G. R. Qldham (1976) "Motivation Through the Design of Work: Test of a Theory", *Organizational Behavior and Human Resources*, (2)16, pp. 250-279.

Hamlet, D. and J. Maybee (2001) The Engineering of Software, Addison Wesley, Boston, MA.

Hannemyr, G. (1999) “Technology and Pleasure: Considering Hacking Constructive”, *First Monday*, 4, pp. 2.
Hazzan, O. and Y. Dubinsky (2007) “Why Software Engineering Programs should teach Agile Software Development”, *SIGSOFT Softw. Eng. Notes*, (2)32, pp. 1–3.
Highsmith, J. (2004) “Agile Project Management: Creating Innovative Products”, Addison-Weasley, USA.
Iacocca Institute. 21 st Century manufacturing strategy, Lehigh University, Bethlehem, PA.
Katz, D. and R. L. Kahn (1966) “The social psychology of organizations”, New York: Wiley.
Keith, R. J. (1960), "The Marketing Revolution" Journal o f Marketing, (January)/24, pp. 35-38.
Kiggundu, M. N. (1981) “Task interdependence and the theory of job design”, *Academy of Management Review,* 6, pp. 499 –508.
Krech, D. (1935) “Measurement of Tension. Paper read at Symposium on Topological Psychology”, *Bryn Mawr College.*
Krech, D., Crutchfield, R. S., and N. Livson (1969) “Elements of psychology (2nd ed.)”, New York, NY: Alfred A. Knopf.
Kuhn, T. (1962) “The Structure of Scientific Revolutions”, University of Chicago Press.

Lawler, E. E. (1973) Motivation in work organizations. Monterey, Calif.: Brooks/Cole.
Leinhardt, G., McCarthy, Y K. and J. Merriman (1995) “Intergrating professional knowledge: The theory of practice and the practice of theory”, *Learning and Instruction,* 5, pp. 401–408.
Levitt, Theodore (1983), “The Marketing Imagination”. New York: The Free Press.

Lewin, K. (1935) “A dynamic theory of personality”. New York, NY: McGraw-Hill.

Lewin, K. (1948) “Resolving Social Conflicts: Selected Papers on Group Dynamics”. Harper and Row, New York.
Mann, J. (1966) “The Role of Project Escalation in Explaining Runaway Information Systems Development Projects: A Field Study”, *Unpublished Ph.D. Dissertation*, Georgia State University.
Maurer,F. and G. Melnik (2005) “What you always wanted to know about agile methods but did not dare to ask”, *in: ICSE ’05: Proceedings of the 27th International Conference on Software Engineering, ACM Press*, New York, NY, USA, pp. 731–732.
Miller, N. E. (1944) “Experimental studies of conflict”, *In J. Hunt (Ed.), Personality and the behavior disorders*, 1, 421-465). New York, NY: Ronald Press.
Monden, Y. (1983) “The Toyota Production System”, Productivity Press, Portland, OR.
Nerur, S., Mahapatra, R. and G. Mangalaraj (2005) “Challenges of migrating to agile methodologies”, Communications of the ACM, pp. 72–78.
Nerur, S. and V. Balijepally (2007) “Theoretical reflections on agile development methodologies: the traditional goal of optimization and control is making way for learning and innovation”, *Communications of the ACM*, (3)50, pp. 79-83.
Ohno, T. (1988) “The Toyota Production System: Beyond Large-Scale Production”, Productivity Press, Portland, OR.
Porter, L. W., Lawler, E. E. and J. R. Hackman (1975) “Behavior in organizations”.. New York: McGraw-Hill.
Porter, L. W., Lawler, E. E. and J.R. Hackman (1975) “Behavior in organizations”. New York: McGraw-Hill.
Rosen, D.E., Schroeder, J.E. and E. F. Purinton. (1998) “Marketing High Tech Products: Lessons in Customer Focus from the Marketplace”, *Academy of Market Science Review*, 98(6).
Schwaber, K. (1995) “Scrum Development Process”, *presented at OOPSLA'95 Workshop on Business Object Design and Implementation.*
Shingo, S. (1981), Study of the Toyota Production Systems, Japan Management Association, Tokyo.
Taylor, F. W. (1947) “Shop management”. (published as part of Scientific management). New York:Harper.
Taylor, F. W. (1911) “The principles of scientific management”, New York: Harper and Bros.
Taylor, F. W. (1947) “Shop management (published as part of Scientific management)”. New York:Harper. (Originally published, 1903).
Tomayko J. and O. Hazzan (2004) “Human Aspects of Software Engineering”, Hingham, MA: Charles River Media.

Trist, E. (1981) “The evolution of socio-technical systems”. Occasional paper.

VII. ABOUT THE AUTHORS

Adarsh K. Kakar is Assistant Professor at Alabama State University. He has over two decades of experience in the Indian software industry and international consulting. His research interests include software development methods and software product management.

Dr. Joanne Hale is a Professor of Management Information Systems at University of Alabama. Her research interests include Information systems development and delivery methodologies, information systems process and quality metrics programs, I.S. project management, context aware computing for disaster response (supported through NSF). She is a winner of The University of Alabama’s National Alumni Association Outstanding Commitment to Teaching Award.

Dr. David Hale is Professor and Director of Management Information Systems at University of Alabama. He is a William White McDonald Family Distinguished Faculty Fellow and Director of The Aging Infrastructure Systems Center of Excellence. His areas of interest include crisis mitigation and response, decision support, enterprise integration, knowledge management, collaborative human-computer problem-solving systems, economics, risk and reliability, and systems development. He has over 50 scholarly and IS professional publications in journals and conference proceedings