Teaching Theories underlying Agile Methods in a Systems Development Course

Adarsh Kumar Kakar
Alabama State University
akakar@alasu.edu

Abstract
Presently Agile methods 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.” [8]. 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 Information Systems. In this article a set of relevant theories that can be included as a module in an Agile methods course curriculum 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.

1. 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 [51]. Thus, knowledge that is useful at work includes the dimensions of both practical knowledge and theoretical understanding. Research on the development of expertise have emphasized that a combination of these dimensions is of fundamental importance [28].

Work-based learning in a laboratory environment by itself is not enough as business settings vary depending on people and context. 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. Providing hands-on experiential knowledge of the process of development is not adequate 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 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 students with a framework that allows them to make sense of their hands-on experience. They serve as a yardstick against which the learnings can be measured 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.

2. Instruction Approach
Agile methods represent a paradigm shift from traditional, plan-based approaches to software development [11]. For many decades research in software engineering was preoccupied with discovering ways to deliver faster, better, and cheaper software. The contributions of academicians include formal methods for measurement and standardization of the software process and a variety of tools and techniques to aid software development. However, in the last decade various suggestions for improvement have come from practitioners culminating in the Agile manifesto (http://agilemanifesto.org/).

The emerging principles and methods such as Extreme programming, Scrum, Crystal methodologies, Dynamic Software development method and Feature Driven Development were together labeled as agile software development. This new approach has had a huge impact on how software is developed worldwide.
The increasing popularity of agile methods has made it imperative that they be taught at university level [18].

Although not explicitly identified or stated, Agile methods are derived from concepts in manufacturing and are an amalgamation of theories from multiple disciplines. Thus the module introducing theories underlying the agile methods could begin with a comparison of how the evolution of Systems Development Methods (SDMs) mirrors those of other manufacturing paradigms. This will provide a context and a fresh perspective to students that will affirm their knowledge of SDMs in general and Agile methods in particular. Students will learn to appreciate how a popular theory or system is wholly upended [26] giving rise to new theories and paradigms. Additionally they will learn to actively reconstruct concepts of their own based on their appreciation of how new concepts emerge.

Building on this foundation the instructor can then explore 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 underlying 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.

3. Evolution of Agile Methods

3.1. 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 [17]. 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 were able to get by with this approach to software development for two reasons. First, no better way had been developed, and second, software was not that complex. 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 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 [17].

3.2. Taylorism and Waterfall

To accomplish this, software developers turned to a more than fifty year old paradigm, called "Scientific Management" [44]. Frederick Winslow Taylor [43, 44] introduced Scientific Management with the aim of controlling 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 suggested by Taylor were aimed at increasing specialization and standardization of work.

As applied to software development, Scientific Management, led to the development of new Taylorist approaches such as the waterfall model and its variants. These methods promoted strong conformance to plan through upfront requirements gathering and systems design, programming standards, code inspections and productivity metrics. They also encourage strict Tayloristic division of labor and the use of role based teams of business analysts, system architects, programmers and testers [33].

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 full requirements definition, followed by a long gap before those requirements are delivered 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 would not go as well as planned.

3.3. 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 full set of user requirements to address the business problem [49]. 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 [36]. 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 [36]. 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.

Although introduced in 2000s, the roots of Agile principles can be traced to the Lean and Agile manufacturing paradigms introduced in the 1980s and 1990s respectively. Lean thinking originated on the shop-floors of Japanese manufacturers and in particular as a result of innovations at Toyota Motor Corporation [35, 38, 42] resulting from a scarcity of resources and intense domestic competition in the Japanese market for automobiles. 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 lean approach focused on creation of value by elimination of waste and represented an alternative model to that of capital-intensive mass production.

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 [20]. 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.

The evolution of software development approaches and the corresponding manufacturing paradigms are summarized in Table 1 below:

<table>
<thead>
<tr>
<th>Manufacturing Paradigms</th>
<th>Software Development Approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Craftmanship</td>
<td>Code and Fix</td>
</tr>
<tr>
<td>Taylorism</td>
<td>Plan-driven approaches such as Waterfall or V Model</td>
</tr>
<tr>
<td>Lean/ Agile Manufacturing</td>
<td>Agile Methods</td>
</tr>
</tbody>
</table>

### 4. Theories

#### 4.1. Process Control Theory

Industrial process control theory specifies two types of process control systems: defined processes and empirical processes [41]. 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 white box systems, as the processes are well defined and understood. Empirical processes are black-box systems. These processes are complex and have unpredictable outcomes. The outcomes are not repeatable for a given set of inputs [41].

Software is considered to be a black box system. There is uncertainty surrounding the business requirements and technology. Applying defined process methodologies such as waterfall methodologies to unpredictable systems might not always work. In a defined process all the unknowns are expected to be solved up-front. However, the assumption made in waterfall methodologies that software development is a well understood process is often not true. Even with highly detailed upfront user interface designs, specifications and plans, the software produced often turns out different from what was originally intended.

Agile approaches consider in-depth effort to fully understand the problem domain wasteful. Uncertainty is not something that can be handled by up-front planning and design but needs processes that are more fluid. Excessive planning is like inventory on the shop floor. Not an asset but a liability.

#### 4.2. Marketing Concept

Agile development approach brings the marketing concept into software engineering by emphasizing the primacy of addressing evolving customer requirements. Keith's [22] article, on the marketing concept illustrates 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 successful
evolution through three managerial phases, production, sales and finally reaching what he calls a marketing control phase.

Customer focus is a core element of the marketing concept [40]. Theodore Levitt’s [29] 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 considered 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 the 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.

4.3. Emergence

Agile methodologists do not believe that a software application can be fully specified up-front as business requirements and technologies change rapidly during the course of a software development project. The true requirements should be allowed to “emerge” over time because what the customer initially thought they wanted gets refined as software develops. Thus in the interest of developing software products that are relevant to the business needs of the organization changing requirements should be welcomed even late in the development cycle. The short development iterations of Agile methods and customer evaluation of working software at each iteration provide the means for business requirements to emerge. The end result is delivering what the customer really wants in the face of constant change in the business environment which the customer operates.

4.4. Job Design Theory (adapted from [47])

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 [44] 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 [48]. Typically, a Taylorist job is highly specialized, and workers are not encouraged to experiment, innovate, or otherwise vary the way that tasks are performed [48].

In the 1970’s, Richard Hackman, Edward Lawler and Greg Oldham suggested that Taylorist job design may not be optimal [13, 15, 27, 39]. Enriched jobs motivate workers by encouraging them to innovate and learn. Motivated workers are known to demonstrate improved task performance and higher productivity. Firms therefore soon put this new approach into practice by redesigning jobs, giving greater autonomy to the employees by adopting self managing teams and creating employee participation programs like the quality circles.

Among the various theories, the Hackman and Oldham (1976) Job Characteristics Model (Figure 1) is one of the most elaborate and widely accepted theories of job design [23]. It can be used effectively to explain the emerging trends and empirical observations in the domain of SDMs.

![Figure 1: The Job Characteristics Theory (JCT)](Reproduced from [15])

Job characteristics theory claims that an individual will be motivated to work when jobs are designed to satisfy three critical psychological states. These include (see Figure 1) the need:

1. for performance feedback
2. for meaningful work
3. to be responsible for work outcomes
These critical psychological states are affected by the five characteristics of job which describe (see Figure 1):

1. Skill Variety. The variety of skills required to carry out work.
2. Task Identity. The clearly visible and identifiable outcome of work.
3. Task Significance. The importance of the work output.
4. Autonomy. The freedom given to the individual to perform the task.
5. Feedback. The information given to the individual about the effectiveness of his performance.

These critical psychological states affect the work performance job satisfaction and employee motivation (see Figure 1). When applied in the context of SDMs, the five characteristics of the model can be used to characterize typical Agile and Taylorist methods of software development (Figure 2).

Taylorist software development methods demand less skill variety to accomplish tasks. Specialized teams are deployed for planning of lifecycle activities such as requirements gathering, design, construction, testing and specialized people handle each of these tasks. The allocation of work specifies not only what is to be done but also how it is to be done and the time allowed for doing it [5]. 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 (http://agilemanifesto.org/). People issues are at the heart of the agile movement [2]. Agile methods are people-centric, recognizing the value competent people and their relationships bring to software development. [36]. The agile team works by placing people physically closer, replacing documents with talking in person and conference rooms, thereby improving the team’s amicability and its sense of community [6].

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 focus on developing working products rather than paper artifacts and components. This enhances task identity and task significance. Big upfront design plans and extensive documentation are of little value to practitioners of agile methods. [37]. 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 [2], 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 [15]:

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

resulting in higher team member morale, satisfaction and productivity. The difference between Agile methods and Tayloristics methods based on JCT is illustrated below:

JCT is useful in explaining the relevance of various agile best practices. For example the benefits of Paired programming can be explained by the rapid feedback programmers working in pairs can provide to each other. Developing working products in rapid iterative cycles enhance task identity and significance of the task completed. Developing whole, meaningful and working products make it easy for developers to identify with the tasks that are fulfilled compared to say requirements specification or prototypes of Taylorist methods. In addition developing working products enable meaningful and timely feedback from the users to the developers. This in turn increases the motivating potential of team members and the resulting work outcomes in software development projects using agile methods. This is in direct contrast to the work approach, such as in Tayloristic methods, where developers are given a specification for parts of the solutions, and do not have full picture of the product this is being developed.
4.5. Approach-Avoidance Theory (Adapted from [49])

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 [3] 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 [32] 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 [4, 7]. 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 [21]. Results from a series of experiments provide support for the completion effect [7, 12]. Specific evidence that the motive to complete a task gets stronger as one gets closer to completion can be found in work by Lewin [30] whose hypothesis was supported in later empirical work [4, 24, 25, 34]. The motivation of individuals and team members increased as the desired goal moved within reach thereby overcoming the costs of persistence and increasing the probability of successful project outcomes.

When the goal appears distant, uncertainty about project outcomes builds up. The cost of persistence, a restraining force, then gains primacy resulting in demotivated individuals and teams, user dissatisfaction and project delays. This is what happened in the classic case of London Stock Exchange which scrapped TAURUS an electronic share-transfer system. Despite work spanning a decade and investments estimated to be as high as 400 million pounds [9, 10]. 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 [9].

Agile Projects 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. At each iteration stakeholders of the project evaluate whether the working product has accomplished the desired goals. 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 gain momentum with increasing motivation, compelling them to continue to invest their efforts and accelerating them towards successful project completion.

4.6 Socio-Technical Systems Perspective

Tavistock group’s study of English coal miner’s in the 1950s was a pioneering attempt to research self-organizing teams from a socio-technical system perspective [46]. Self-Organizing teams are autonomous groups that respond to every day learning in making decisions. They exercise a democratic system of control called concertive control [31]. This is in stark contrast to the bureaucratic control which relies on hierarchy and rewards compliance [1]. Self-organizing team’s use of concertive control enhances the ability of the team to respond to rapidly changing business conditions [31].

Agile methodologies give development team the autonomy to self organize to determine the best way to get the job done. Team members neither have predetermined roles nor are required to execute outdated plans. Managers place a lot of trust and confidence in the entire team and promote face-to-face conversations among developers and between developers and the business people or customers. Team member reflect on what went right and how to become more effective during frequent review meetings and adjust their behavior accordingly.

5. Contribution and Conclusion

This article responds to a call for augmenting our knowledge of the theoretical bases of Agile methods [16, 45]. Software development is a complicated task consisting of technical, social and cognitive dimensions. Accordingly, the academy has a significant and non-trivial role in the education of future software developers toward this multifaceted challenge [50]. 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 this study suggests an outline of the theoretical component which should become an integral part of any agile system development course. The theoretical underpinnings of agile principles discussed in this article are summarized in Table 2.
<table>
<thead>
<tr>
<th>Agile Principles</th>
<th>Theories and Paradigms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.</strong></td>
<td>Agile Manufacturing (Early Delivery), JCT (Task Significance – Valuable Software), Marketing concept (Customer Satisfaction)</td>
</tr>
<tr>
<td><strong>Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.</strong></td>
<td>Agile manufacturing (Flexibility – welcome change), Marketing concept (Respond to changing customer requirements)</td>
</tr>
<tr>
<td><strong>Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.</strong></td>
<td>Agile manufacturing (Speed), JCT (User Feedback), Completion Effect (Working Software)</td>
</tr>
<tr>
<td><strong>Business people and developers must work together daily throughout the project.</strong></td>
<td>Marketing concept (Customer Collaboration)</td>
</tr>
<tr>
<td><strong>Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.</strong></td>
<td>JCT (Autonomy), Socio-Technical perspective (Empowered Teams)</td>
</tr>
<tr>
<td><strong>The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.</strong></td>
<td>JCT (Task Identity), Completion Effect (Working software)</td>
</tr>
<tr>
<td><strong>Working software is the primary measure of progress.</strong></td>
<td>JCT (Task Identity), Completion Effect (Working software)</td>
</tr>
<tr>
<td><strong>Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.</strong></td>
<td>Lean Manufacturing (Focus on Quality)</td>
</tr>
<tr>
<td><strong>Continuous attention to technical excellence and good design enhances agility.</strong></td>
<td>Lean Manufacturing (Focus on Quality)</td>
</tr>
</tbody>
</table>

6. References


[20] Iacocca Institute. 21st Century manufacturing strategy, Lehigh University, Bethlehem, PA.


