Migrating Agile Methods to Standardized Development Practice

Situated process and quality frameworks offer a way to resolve the tensions that arise when introducing agile methods into standardized software development engineering.

Mark Lycett
Robert D. Macredie
Chaitali Patel
Ray J. Paul
Brunel University

Modern business organizations are *emergent*—they reside in a state of continual process change.[1] Globalization, deregulation, increased competition, mergers and acquisitions, and the like all reveal organizations in transition, adapting to a continuously changing business environment. In this dynamic context, however, organizations must still project quality of product, service, and process to gain market presence and competitive edge. Significant organizational tension arises as the stability that underpins notions of quality control is overlaid on environments in flux.

In software development, this tension is most obvious in the difference between standardized engineering approaches and more recent agile methods. Engineering approaches tend toward explicitly defined processes that can be standardized both within and across organizations. The significant industry investment in engineering approaches is evident in standards such as the International Organization for Standardization's ISO 9000 series, the Software Engineering Institute's Capability Maturity Model (CMM), and a number of industry-led initiatives such as the Object Management Group's Unified Modeling Language and the Rational Unified Process (RUP) from Rational Software.

However, the agile development movement has begun to question the philosophy behind engineering initiatives. Agile development emphasizes the human or crafted aspects of software development over the engineering aspects—individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan.

The "Industrial Context: AgCo Legacy Migration" sidebar introduces our work with a large international systems integrator, which we pseudonymously call "AgCo." In this work, we found engineering and agile perspectives to be equally important to modern software development. We developed a framework approach to implementing agile values and principles in the context of AgCo's organizational mandate for the consistent use of RUP and CMM.

BACKGROUND

The rise of capitalism, modern bureaucracy, and scientific management has propelled the creation of managed, repeatable, and quantifiable organizational processes.[2] In general, the power of rational systems has been proved in the context of market competition and is well evidenced in the success of the "manufacturing" approach to production.

Software crisis

In 1968 and 1969, the North Atlantic Treaty

0018-9162/03/$17.00 © 2003 IEEE

Industrial Context: AgCo Legacy Migration

AgCo (a pseudonym) was originally a mainframe manufacturer. Since the mid-1990s, however, the company has moved its strategic focus toward information management. This focus includes industrial and organizational expertise as well as the application and support of information technology. Thus, the company offers its services in the areas of systems endurance, business transformation, business enhancement, and systems replacement. Although these services are available across a range of industrial sectors, financial services provide a major revenue area and one in which the company has a considerable history.

AgCo has several legacy products catering to the financial services industry. The products were developed in a fourth-generation language. Perceived customer demand and technology advances motivated a decision to migrate three legacy products toward a service-based environment. AgCo wanted to increase product flexibility and scalability as well as component reusability, thereby supporting system tailoring and growth to suit changing contexts. It expected to do so with less overhead than traditionally developed systems impose.

Legacy migration was part of a larger organizational strategy aimed at standardizing organizational development approaches. AgCo has adopted the Rational Unified Process (www.rational.com/products/rup/) for software development and is working toward quality certification via the Capability Maturity Model Integration (www.sei.cmu.edu/cmmi).

RUP uses an iterative, incremental approach to software development, offering a set of common processes to improve communication and create a common understanding of all necessary development tasks, responsibilities, and artifacts. CMMI provides models for organizational process improvement in the areas of software, system engineering, and integrated product and process development. In addition to providing a "staged" model for process improvement and audit, CMMI offers a continuous representation that allows different organizational processes to improve capabilities at their own pace.

AgCo adopted both RUP and CMMI with an engineering philosophy in mind. To develop a means of legacy migration, it needed to institutionalize RUP and align the migration with it. To achieve and demonstrate process maturity, it needed to institutionalize CMMI as well.

Clearly, institutionalization involves significant, ongoing organizational change, but it also implies that "one size fits all." To some extent, this accurately reflected management thinking, but it does not filter through to the reality of development practice, which commonly adapts processes and so on. This disconnect signals an opportunity for introducing agile software development.

Organization held two conferences to discuss the "software crisis." The conference proceedings document the perceived need for rational systems in software development.[3,4] Perceived problems included the increasing size and complexity of software systems, difficulties in estimating costs and managing large numbers of people, shortages of skilled software professionals, emphasis on coding at the expense of design and testing, and poor documentation.

Key conference recommendations sought to standardize the software development process with emphasis on quality, costs, and development practices. In keeping with a rational system approach, proponents expected the process to improve quality at the same time it reduced costs and increased productivity.

Software development has since come to rely on a methodical approach. Comprehensive development reviews testify to this reliance, which is also evident in an identifiable orthodoxy in information systems research.[5] This orthodoxy structures the system development process through a purposeful framework that organizes and regulates activities and ensures their comprehensiveness.

From crisis through depression to agility

Because software is so woven into the fabric of modern business, it is in many ways pointless to argue that the engineering approach has failed. However, the manufacturing ideal of software production remains unfulfilled, and significant evidence indicates that many of the software crisis problems remain with us today.

Results of a recent study, for example, showed that 80 to 90 percent of software does not meet performance goals, 80 percent of systems are delivered late and over budget, about 40 percent of developments fail or are abandoned, less than 25 percent of systems properly integrate business and technology objectives, and only 10 to 20 percent meet their success criteria.[6]

A compelling perspective on these enduring problems argues that software development process is not a product so much as a communication medium.[7] The process must translate tacit, evolutionary, and often undefined knowledge into digital form. The translation involves not just basic grammar, syntax, and rules but also issues of meaning and intent that are both contextual and subjective.[2] In this view, software development is a knowledge-acquiring as well as a product-producing activity, and its structure must address a communication medium's demands in good part.

The agile approach implicitly embodies this view. It strongly favors human communication and collaboration over defined and repeatable activities as mechanisms for developing quality software. In practice, the difficulties that many organizations face with the agile approach reside, first, in the cultural change necessary to convince management of its benefits and, second, in its feasibility within the increasingly global market forces, regulatory practices, and government oversight that fuel standardization.

To reduce the disjunction between software development management, theory, and practice that has remained unresolved since the NATO con-

ferences,[2] we must examine the needs that emerge from each perspective.

Process theories and realities

In engineering development, the demands on software development are relatively well articulated. The perspective assumes that requirements can be fairly well defined and that the development process concentrates on moving software process control from producers to managers. Three principles underlie this view:

- *Formalism*—seeking a universal approach to a wide range of problem situations. An abstract set of processes, activities, and so on transform inputs into outputs to deduce a repeatable solution.
- *External knowledge capture*—capturing and standardizing the development process to support a learning paradigm of inert knowledge transfer as a basis for coordination, communication, and training.
- *Economics*—using the division of process and labor to rationalize cost and effort. By establishing a purposeful framework of component activities, management can eliminate the activities that appear to be redundant, irrational, and counterproductive. It can also partition the process, allowing task specialization and pay rates differentiated for particular skills.

Several theories influence this approach.[8] A *transformation* view of process, which decomposes a problem into sequential activities that change inputs to outputs, has dominated production thinking for many decades.

The management tasks of planning, execution, and control all have underlying theories. *Planning* assumes that the organization consists of separate management and production elements—that "thinking" can be separated from "doing" and that producing plans is synonymous with action. *Execution* provides the interface between plan and work, dispatching authorization to proceed. *Control* uses a thermostatic model that monitors variances between standard and actual values and corrects the process accordingly.

The realities of emergent organizations, however, are disjoint from these theories, which have failed to solve the software crisis problems.

In agile development, solutions evolve collectively over time. The process concentrates on consolidating control with the producers, thus implying a coordination role for management. The principles underlying this view contrast with the engineering approach:[9,10]

- *Context awareness*—varying the approach based on a collective understanding of contextual needs and delivering working software frequently as the principal check on quality and progress.
- *Individual excellence*—emphasizing the human aspects of skill development and motivation, mandating management trust, and exploiting pride in the job.
- *Peer-based knowledge capture*—using pairing and periodic group reflection as mechanisms for explicit collective learning and emphasizing face-to-face communication as the principal means of knowledge transfer—explicit and implicit—between developers and customers.
- *Minimalist methodology*—concentrating only on effort that is judged to be necessary and sufficient in a particular situation.

The agile development process implies a coordination role for management.

From a production perspective, agile development is more akin to *value generation* than to transformation. Value generation aims to provide the best possible value from the customer's perspective.[11] It accepts that customer requirements are neither necessarily available nor well understood and that allocating requirements to different development activities is a difficult problem.

From a management perspective, *management-as-organizing*[12] supersedes planning, executing, and controlling. In this view, management involves the design, coordination, and enabling of activities and is primarily concerned with shaping the physical, political, and cultural environment for action. Importantly, agile development views action as inherently *situated*—it is never adequately captured in a plan.

RESOLVING TENSIONS

Agile development is based on theory more appropriate to emergent organizations, but it faces a significant barrier to implementation: the resistance of large, mature organizations to wholesale change in working practice. However, while theories may differ, the broad goals of engineering and agile development are implicitly the same: produce the intended software artifact, observe internal needs such as minimizing cost, and fulfill external customer-related needs such as quality and flexibility. The congruence in these broad goals provides a lever for migrating organizations across the theoretical divide.

A process generic enough to deal with any situation is too high-level to be practical.

However, two practicalities arise from attempting to introduce agility while continuing to project product, service, and process quality while managing both a market presence and a competitive edge.

Situated processes

The primary tension that agile principles and values impose on development practice is in balancing between implementing repeatable processes and allowing for the nuances in a particular development context.

From an engineering development perspective, management expects predictable processes to achieve its primary goal of quality product development at minimal cost. But uncertainty and contextual differences are a fact of life in software development. While it may appear easier to plan, execute, and control development via globally applied standard processes, management is not useful if it does not reflect project realities. Further, a process generic enough to deal with any situation is too high-level to be practical. It must be tailored, which requires substantial up-front effort. Many reuse efforts attest to this observation, yielding neither use nor reuse.

Agile software development accepts these difficulties, adopting collaborative principles and values that engender informal trust and interaction; a chaordic perspective that sees complete order as a fallacy; and a "barely sufficient" methodology[9] that reflects a "less is more" philosophy.

To management, these elements shout risk and, furthermore, risk that increases significantly in relation to project size. The contractual nature of much large-scale software development makes it easy to understand why production and management theories are what they are. Developers at AgCo, for example, rarely have contact with clients—other parts of the organization act as gatekeepers in this regard.

Rational Unified Process. Given AgCo's adoption of RUP and pursuit of CMM quality certification, we proposed using RUP for a learning framework that would educate the organization in agile principles while mitigating risk.

At a senior management level, we first intended to standardize on a set of activities, artifacts, workflows, and roles for each RUP discipline across three legacy migration projects. This approach seemingly offered easier rollout and monitoring. However, a significant number of RUP activities and actifacts soon proved to be unsuitable to the development context of legacy migration.

At a tactical management level, we tried mapping RUP activities and artifacts to the existing AgCo methodology to provide some consistency of understanding. However, this approach also became problematic. Even though we completed a nominal mapping at the activity level, RUP's iterative and incremental aspect was new to AgCo, which had a waterfall mentality. A simplistic mapping of RUP standards to existing company methods only repeated difficulties that inhere in the engineering approach.

Explicit tailoring. Accounting for a more situational view, the most culturally acceptable solution was to translate implicit tailoring to explicit tailoring. This meant finding a level at which we could broadly reuse processes, activities, artifacts, and the like. We thus sought to migrate to a more agile approach by

- Factoring a core set of artifacts judged to be necessary—but not necessarily sufficient—for a wide range of development. These artifacts provided a means for working back to the core generic processes and activities. We took artifacts as a starting point because AgCo management saw them as the elements providing the most value.
- Identifying a set of candidate patterns judged to cover (a) the core types of development—*legacy migration* being one type—and (b) the contextual application of activities, artifacts, and guidelines. Patterns were used as they provide repeatable solutions to a given contextual problem. Core patterns provide a means of configuring a set of context patterns to meet a development level need.
- Examining each pattern's additional artifacts for those considered both necessary and barely sufficient.
- Producing a skeletal decision framework for selecting patterns from four characteristic sets: *project elements*, such as scope, scale, and novelty; *product elements*, such as technology base and complexity; *team elements*, such as expertise, size, and work location; and *organizational elements*, such as structure and cultural mentality. These characteristic sets can expand or evolve with the repository of good working practice/pattern catalogs.

The framework implements a "learn by doing" philosophy and an evolving repository of good working practice. The practice is embodied in patterns documented by lightweight development

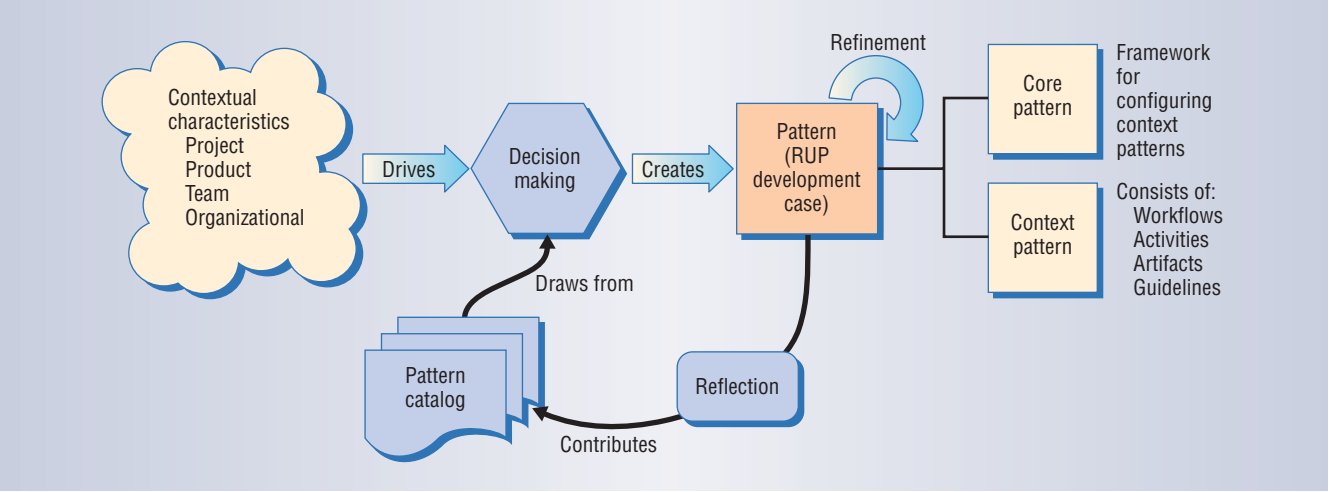

Figure 1. Situated process framework. Patterns are developed via a situated examination of contextual characteristics, which results in a pattern catalog of RUP workflows, activities, artifacts, and the like. Patterns are expressed as an RUP development case. The development case is reviewed and refined as the project goes through iterations and is housed in the context pattern repository of good working practice.

cases. The cases, in turn, are based on RUP artifacts that capture configuration and rationale on necessary and sufficient aspects of a development pattern. The overall framework is not prescriptive per se. Rather, it is an aid in tailoring the process to context and in educating the people involved. Consequently, developers actively revisit the case in each development iteration.

Visibility. Figure 1 shows how the situated process framework functions. In practice, making the decision process visible is extremely valuable because it forces management to actively determine whether an activity or artifact is both necessary and sufficient.

Table 1 describes how the framework applies in the context of agile principles and practice. In broad terms, the situated process framework aims to orient users toward adopting agile development in everyday practice by suggesting suitable techniques for collaboration, interaction, and communication. For example, observing a pattern's evolution can suggest new or more effective ways to apply the pattern in specific situations. It can also help in determining how often the process must deliver software to form effective feedback and trust loops as well as what forms and frequency of communication are necessary for effective delivery.

Table 1. Embedding agility into the situated process framework.

Agile principles	Situated process framework
Different projects need different processes.	Tailoring a process to meet a project's contextual needs
In methodology, "less is more."	Selecting patterns, activities, and artifacts based on the "barely sufficient" criterion
Deliver working software frequently and establish feedback loops.	Adopting an incremental and iterative approach to enforce iteration of all software development phases, including build
Enhance interaction and communication.	Suggesting interaction and communication techniques suitable to a project's contextual characteristics
Empower individuals to make decisions.	Enabling individuals to further tailor selected patterns
Learn by doing.	Capturing good working practice to enrich the framework

Situated quality

A second tension that agility imposes on engineering practice relates to the perception of quality. From an engineering perspective, quality is strongly linked to the use of standards and the subsequent certification against those standards. Most quality standards, such as ISO 9000 and CMM, function under the "say what you do, then do what you say you do" philosophy. In practice, certification requires an audit of both the "saying" and "doing" and assumes that process quality will equate to product quality.

The audit mentality mandates tangible output and tends toward common development difficulties such as excessive documentation, reduced communication-oriented activity, and a growing disjunction between the working code and documentation.

An agile perspective puts more emphasis on product quality. Process quality can lead to product quality, but a good process is not necessarily standardized or repeatable. The agile perspective sees a key distinction between consistency and repeatability. Emergent situations never give rise to repeatability because the development context—information, constraints, politics, and so on—is always different.[10]

Outcome consistency, however, is more likely achievable when project aspects equate with similar experience and when the experience is oriented toward reflective practice as opposed to "following a process." However, in keeping with the engi-

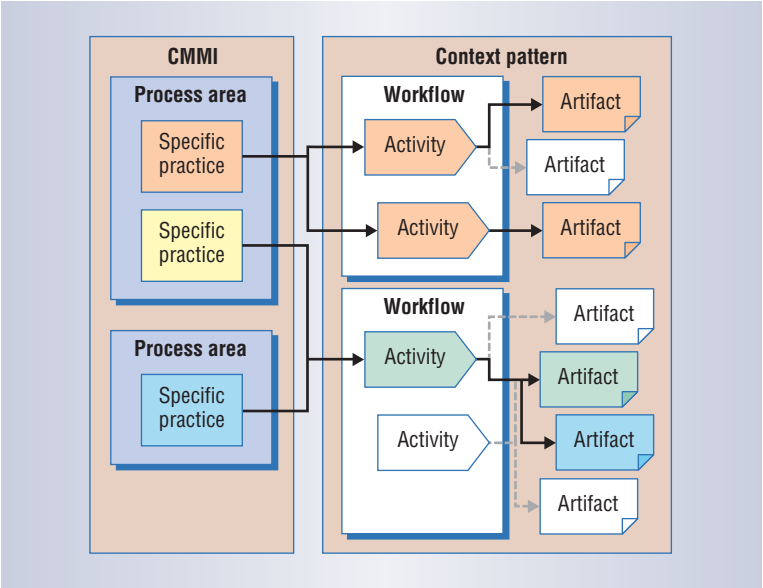

Figure 2. Mapping Capability Maturity Model Integration practices to process pattern elements. We map the CMMI specific practices within each process area to context patterns on many different levels: RUP workflows, workflow details, activities, artifacts, and guidelines.

neering perspective, customer trust currently rests in the software process infrastructure.

The institutionalization of quality was important to AgCo from both the practical development and marketing perspectives. However, the organizational aim of mapping CMM Integration practices to RUP artifacts as directly as possible created the impression of a "one size fits all" process that did not work. We solved the immediate difficulty by

- adopting the pattern-based framework to supplant repeatability with consistency, while still providing the audit trail necessary for assessment; and
- mapping CMMI practices to candidate development patterns and accepting that combinations of RUP workflows, activities, artifacts, and so on can participate in more than one CMMI practice.

Figure 2 illustrates this mapping.

This approach does not violate the "say what you do, do what you say you do" principle, but it does seek to replace the weak notion of repeatability with a strong notion of consistency in approach, decision making, and expected product quality. It has important implications for certification—specifically, an audit of continuously developing capability, which in turn suggests continuous CMMI implementation.

In addition, the approach's tailored nature suggests that process capabilities will be certified at the level of "Defined," which explicitly acknowledges the value of tailoring and configuration.

Organizations must derive development patterns and the necessary and barely sufficient processes, activities, and artifacts from their experience over time. Our learning framework allows mature organizations to migrate from an engineering view of development to a more agile view.

The argument that software development has a strong communication element means that an entirely rational process model will always have limited applicability. Accepting this problem is a prerequisite for addressing it. At the same time, large and mature organizations are unlikely to adopt a big-bang approach to agility. Management sees the associated risks as too great.

Despite a clear rationale for a migratory approach, it is not without difficulties. First, building the pattern repository takes time and effort. Further, the repository's evolution requires some form of configuration management system. Finally, the framework and configurations pose complex maintenance issues.

We propose the application of agile thinking to manage these issues. Agile process does not fly in the face of engineering practice. If used thoughtfully, it provides a clear mandate for making engineering practice lean and well focused. For the approach to be truly successful, however, organizations must grasp the opportunity to reintegrate software development management, theory, and practice. The effects of disjunction have been evident long enough. ■

References

1. D.P. Truex, R. Baskerville, and H. Klein, "Growing Systems in Emergent Organizations," *Comm. ACM*, vol. 42, no. 8, 1999, pp. 117-123.
2. K. Eischen, "Software Development: An Outsider's View," *Computer*, May 2002, pp. 36-44.
3. P. Naur and B. Randell, eds., "Software Engineering: Report of a Conference Sponsored by the NATO Science Committee," 1968; http://homepages.cs.ncl.ac.uk/brian.randell/NATO/.
4. B. Randell and J.N. Buxton, eds., "Software Engineering Techniques: Report of a Conference Sponsored by the NATO Science Committee," 1969; http://homepages.cs.ncl.ac.uk/brian.randell/NATO/.
5. R. Hirschheim, H.K. Klein, and K. Lyytinen, *Information Systems Development and Data Modeling: Conceptual and Philosophical Foundations*, Cambridge Univ. Press, 1995.
6. C. Clegg et al., *The Performance of Information Technology and the Role of Human and Organiza-*

tional Factors, tech. report, Economic and Social Research Council, UK, 1996.
7. P.G. Armour, "The Case for a New Business Model," *Comm. ACM*, vol. 43, no. 8, 2000, pp. 19-22.
8. L. Koskela and G. Howell, "The Underlying Theory of Project Management Is Obsolete," *Proc. PMI Research Conf.*, Project Management Institute, 2002.
9. A. Cockburn, *Agile Software Development: Software Through People*, Addison-Wesley, 2002.
10. J. Highsmith, *Agile Software Development Ecosystems*, Addison-Wesley, 2002.
11. H.E. Cook, *Product Management: Value, Quality, Cost, Price, Profit and Organization*, Chapman and Hall, 1997.
12. R.B. Johnston and M. Brennan, "Planning or Organizing: The Implications of Theories of Activity for Management of Operations," *Omega*, vol. 24, no. 4, 1996, pp. 367-384.

Mark Lycett *is a research project manager and lecturer for the Department of Information Systems and Computing at Brunel University. His research interests include all aspects of software development, with emphasis on agile, adaptive, and evolutionary systems. Lycett received a PhD in information systems from Brunel University and is a member of the IEEE, the ACM, and the Project Management Institute. Contact him at Mark.Lycett@ brunel.ac.uk.*

Robert D. Macredie *is a professor of interactive systems in the Department of Information Systems and Computing at Brunel University. His research is broadly grounded in information systems development, particularly the integration of people, process, and technology. Macredie received a PhD in computer science from the University of Hull and is a member of the IEEE, the ACM, and the British Computer Society. Contact him at Robert.Macredie@brunel.ac.uk.*

Chaitali Patel *is a research assistant in the Department of Information Systems and Computing at Brunel University. Her research interests are in agile systems development, particularly method and process. Patel received an MSc in distributed information systems from Brunel University. Contact her at Chaitali.Patel@brunel.ac.uk.*

Ray J. Paul *is a professor of simulation modeling in the Department of Information Systems and Computing at Brunel University. His research is broadly grounded in information systems development, with an emphasis on adaptive and evolutionary systems. Paul received a PhD in operational research from the University of Hull. He is a member of the Operational Research Society and a fellow of the British Computer Society. Contact him at Ray.Paul@brunel.ac.uk.*