Agility at Scale

Economic Governance, Measured Improvement, and Disciplined Delivery

Alan W. Brown — Surrey Business School, University of Surrey, UK
Scott Ambler — Scott Ambler + Associates
Walker Royce — IBM

*Abstract–***Agility without discipline cannot scale, and discipline without agility cannot compete. Agile methods are now mainstream. Software enterprises are adopting these practices in broad, comprehensive delivery contexts. There have been many successes, and there have been disappointments. IBM's framework for achieving agility at scale is based on hundreds of successful deployments and dozens of disappointing experiences in accelerating software delivery cycles within large-scale organizations. Our collective know-how points to three key principles to deliver measured improvements in agility with high confidence:**

1. **Steer using economic governance**
2. **Measure incremental improvements honestly**
3. **Empower teams with disciplined agile delivery**

This paper elaborates these three principles and presents practical recommendations for achieving improved agility in large-scale software delivery enterprises.

*Index Terms–***Software process improvement; accelerating software delivery; agile development; integration first; economic governance; measured improvement; steering leadership; disciplined agile delivery.**

I. Introduction

The principles of agile software development are now more than 10 years old [1]. Achieving agility at scale, however, demands process optimization among groups of stakeholders as well as applying agile principles in context. Many practical ideas have been described, piloted, practiced, and evolved. Agile practices are now neither novel nor extreme. However, applying agile approaches at large scale remains fraught with complexity, just as it has been for past eras of significant software process improvements.

Most agile adoption efforts begin with small teams of software developers introducing agile practices. These practices are not part of any formal organizational transformation; they are simply team efforts to deliver good software effectively. When the results are good, other small software development teams quickly adopt similar practices.

Despite early enthusiasm and grassroots success for agile practices, agile proponents face barriers to broader acceptance when they come up against organizational challenges in scaling, governing, and institutionalizing. All too often, the organization rushes into an enterprise-wide agile improvement program and then confronts disappointment, skepticism, and a shortfall in quantified results.

In our experience, both within IBM and in the broader industry, many change-averse groups perceive agile processes as the results of shoddy planning, poor management, and chaotic delivery schedules. Consequently, these organizations become increasingly polarized. The process *progressives* are advocating dynamic planning, steering leadership, rapid delivery cycles, integration-first milestones, and empowered teams. They demand change. The process *traditionalists* defend their reliance on plan-and-track leadership, overly precise plans and specifications, integration-later milestones, and hierarchical teams. They resist change.

As grassroots success starts to spread, the supporting constituencies (the resource managers, financial managers, sellers, and other support teams in a software business) find themselves to be outside resistors. They exaggerate the challenges that agile approaches inflict on their traditional mindset and techniques. For example, they may observe less precise up-front commitments to resources, less long-term planning, new progress/quality measures that don't align with company baselines, and so forth. They see new techniques as problems for their localized turf, not as a source of possible benefits for the larger business and organizational efficiency.

In the worst cases, the result of these misaligned teams is that agile adoption stalls, stifled by middle management — project managers, business analysts, contract negotiators, and others. These supporting stakeholders do not feel empowered by agile approaches; they feel threatened and out of control.

IBM has worked with hundreds of clients who are introducing modern agile techniques into their enterprise-scale software delivery approaches. We have found that success in achieving agility at enterprise scale requires three foundational principles:

1. *Economic governance.* An objective economic foundation for planning, decision-making, and progress reporting that resolves uncertainties earlier and unifies constituencies on a shared set of expected target outcomes.
2. *Measured improvement.* A more honest approach to measuring progress and quality outcomes by quantifying change trends in the code and test base.
3. *Disciplined agile delivery.* A process decision framework that defines a set of practices, measures, and know-how. Discipline ensures progress is goal-driven, using a hybrid approach to IT solution delivery.

First, we base the framework on an economic model for software delivery that recognizes the variations in delivery methods that result from using a mix of waterfall, iterative, and agile techniques. This allows us to reason about where, how, and when agile approaches can provide real benefits to an enterprise organization. Second, we introduce an incremental, measured approach to agile software delivery improvement. This forms the context for scalable deployment of agile practices as part of a governed organizational transformation. Third, we extend existing agile software development methods such as Scrum with a set of disciplined practices for software delivery in a larger scale, enterprise context. This creates a workable method that provides more freedom to practitioners through less overhead work and more control to management stakeholders through better measurement of progress and quality insight.

II. Relationship to Previous Work

IBM's framework for achieving agility at scale is founded on a long history of exploration and experience in delivering quality enterprise software. As far back as the NATO reports [2], [3] that proposed the term "software engineering" and the original "spiral" and "iterative" models of software development [4], [5], the industry has sought scalable approaches to enterprise software delivery. The culmination of that work has been in improvement frameworks such as the Capability Maturity Model Integration (CMMI) [6], [7], which focuses on repeatability and consistency of system and software delivery. In practice, "maturity" is often achieved at the expense of innovation and flexibility in the delivery processes.

The pressures in many domains to deliver new features more quickly led many teams to focus on agile techniques as both a reaction and a complement to that previous work. In the past decade, work on agile approaches has matured to encompass a very wide variety of software development techniques. While these methods were initially targeted at software development in small, collocated teams, their broader popularity has led to greater emphasis on scaling agile adoption in more complex environments. We can divide that work into four main threads:

1. Several frameworks for scaling agile adoption have been proposed [8], [9], [10]. These frameworks define a set of dimensions through which the challenges of applying agile techniques can be described, measured, and systematically addressed. These dimensions include team size and geographic distribution, as well as regulatory compliance, domain complexity, technical complexity, and organizational distribution. Staged improvement programs can then be executed while addressing those dimensions.
2. Through experiences with specific agile methods like Scrum, scaling extensions and improvements have been proposed [11], [12]. These typically look at the implications of using specific methods in larger, distributed team contexts.
3. Documented case studies of agile approaches have highlighted the practicalities of adapting them to meet large-scale software delivery team circumstances. Interesting examples include IBM [13], Yahoo! [14], and TCS [15]. Each of these case studies emphasizes the organizational, cultural, and financial challenges that must be overcome in specific situations.
4. Extending agile principles to broader areas of the system lifecycle, such as requirements definition [16] and project planning [17], and to more controversial topics such as whether agile practices can be blended with more traditional methods to create hybrid approaches such as "water-scrum-fall" [18] and "wagile" [19]. Many of these lessons are well summarized in Sutherland´s 10-year retrospective on agile adoption [20].

Accelerating software delivery at an enterprise scale is being pursued by almost every business that depends on their software capability as a market discriminator. However, we still lack adequate empirical evidence and well-documented case studies. Our hope is that this paper will provide some target patterns for others to validate or challenge with their own measured improvement know-how.

III. Improving Software Economics

Empirical cost estimation models (like COCOMO II, SEER, QSM, Slim, and others) can estimate resources to within 25 to 30 % for most software projects. This level of unpredictability requires that software governance techniques accommodate high levels of uncertainty. These cost models include dozens of parameters and techniques for estimating a wide variety of software development projects. However, they can be simplified into a function of four basic parameters [21]:

$$\textbf{\textit{Resources}} = \textbf{\textit{Complexity}}^{\textbf{\textit{Agility}}} * \textbf{\textit{Collaboration}} * \textbf{\textit{Automation}}$$

1. *Complexity*. The complexity of software is typically quantified in units of human-generated code (product and test and supporting code) and its quality. Quantities may be assessed in lines of source code, function points, use-case points, or other measures. Qualities like performance, reuse, reliability, and feature richness are also captured in the complexity value. Simpler and more straightforward applications will result in a lower complexity value.
2. *Agility*. This process exponent typically varies in the range 1.0 to 1.25 and characterizes the governance methods, techniques, maturity, appropriateness, and effectiveness in converging on wins for all stakeholders. Better agility (better processes with optimized practices for the project at hand) will result in a lower exponent.
3. *Collaboration*. This parameter captures the skills, experience, motivations, and know-how of the team, along with its ability to collaborate toward well-understood and shared goals. More effective teams will result in a lower multiplier.

4. *Automation*. This parameter captures the extent of process automation, process enactment, instrumentation, and team synchronization. Better tools and instrumentation will result in a lower multiplier.

The mathematical form of this equation, the empirical data in the models, and their practical application across thousands of industry projects indicate that these four parameters are in priority order of potential economic leverage. A 10% reduction in complexity is worth more than a 10% improvement in the process, which is worth more than a 10% more capable team, which is worth more than a 10% increase in automation.

Figure 1 shows ranges and probability distributions of expected improvements based on our experience. Actual targets and outcomes are highly dependent on the specific context. The more significant improvements, like systematic reduction in complexity and major process transformations, also require more significant investments in time and resources. Smaller, incremental process improvements, skill improvements, and automation improvements targeted at projects or individuals are more predictable.

Although tools are important, we have learned from field experience that there is much more economic leverage in reducing complexity, improving process agility, and improving collaboration. Investments in object-oriented languages, Unified Modeling Language, service-oriented architectures, model-driven development, and reuse can help design teams build million-line programs with only diseconomies of scale, instill good practices, codify patterns of success, and avoid patterns of failure. An open thousands of lines of human-generated code. Investments in iterative development and agile methods help to reduce the collaboration platform for enhanced collaboration improves team productivity through integrated change management, agile project management, , and automated instrumentation.

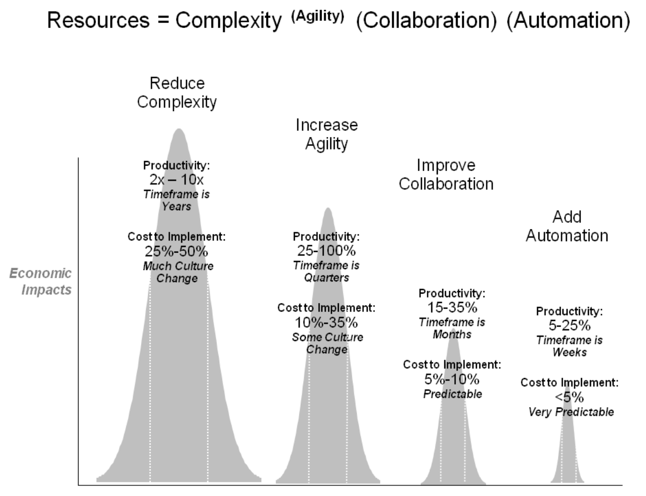

Figure 1. Ranges of productivity improvement

Figure 1 illustrates the expected costs and productivity improvements based on IBM's experience and research. We continue to synthesize this experience into value traceability trees, metrics patterns, benchmarks of performance, and instrumentation tools to provide a closed-loop feedback-control system for improved insight and management.

Past measurement approaches [21], [22] suggest three genres of software governance. We briefly explore these and offer some evolutionary context for why transformations to better measurement are needed.

1. **Engineering governance.** The waterfall model [23] is practiced by the majority of software development teams today because it derives from the predominant legacy culture of traditional engineering governance. Measures typically include (dishonest) earned value based on activity/milestone completion, code/test production, requirements-to-code traceability, inspection coverage, and variances between static plans and actual expenditures. An *engineering orientation* focuses on static targets, planned activities, and deterministic predictions to drive the development process. Waterfall management is simple, but it is overly simplistic for software where uncertainties dominate a project's timeline.
2. **Hybrid governance.** About 20 to 30 % of system and software development teams practice iterative development techniques [24], where architectures are constructed first and evolved through a sequence of executable releases. Measures typically include more honest earned value based on release content, release quality, prioritized risk lists, change management trends, and variances between dynamically changing plans and actual expenditures. Although iterative development is more complex to manage, therefore requiring more project management savvy, it is more frequently successful.
3. **Economic governance.** An *economic orientation* steers project priorities and results based on the dynamically changing predictions of the probable outcomes of the development process. As many as 45% of industry teams claim to practice agile or lean software delivery techniques [25], [26], [27]. Project teams focus on early reduction of uncertainty, continuous integration, asset-based development, and increased stakeholder interaction, and they use smarter, collaborative environments. Measurements include demonstrable user stories, uncertainty reduction (trends in the variance of estimates to complete), defect trends, change trends, and rework trends. Management savvy, combined with meaningful measurement and instrumentation, results in higher levels of agility and improved predictability of outcomes.

Projects that have transitioned to a more agile steering leadership style based on effective measurement can optimize scope, design, and plans, reducing unnecessary scrap and rework. *Steering* implies active management involvement and frequent course corrections to produce better results. Effective steering eliminates uncertainties earlier and significantly improves the probability of win-win outcomes for all stakeholders. Scrap and rework rates are not driven to zero, but to a level that corresponds to healthy discovery, experimentation, and production commensurate with resolving the uncertainty of the product being developed. We estimate this to represent about 15 to 20% of total effort spent in reworking the code and test base, as opposed to the 40% of effort more typical with conventional governance [24].

Most software organizations are struggling to transform their lifecycle model from a development focus to a delivery focus and to improve the economic outcomes of software development. This subtle distinction in wording represents a dramatic change in the principles that are driving the management philosophy and the governance models.

A *software development orientation* focuses on the various activities required in the development process, while a *software delivery orientation* focuses on the results of that process. Organizations that have successfully made this transition have recognized that engineering discipline is trumped by economics discipline in most software-intensive endeavors. Table 1 provides a few differentiating indicators of a successful transformation from conventional engineering governance to more economically driven governance.

To transform successfully from conventional engineering governance to more agile economic governance requires a significant cultural transformation that is best achieved through the pursuit of two major change themes: 1) integration testing should precede unit testing, and 2) agility and control must be complementary.

TABLE 1. TRADITIONAL GOVERNANCE VS. ECONOMIC GOVERNANCE

Software Development Traditional Governance	**Software Delivery Economic Governance**
Distinct development phases	Continuously evolving systems
Distinct handoff from development team to maintenance team	Common process, platform, and team for development and maintenance
Distinct and sequential activities requirements to design to code to test	Sequence of usable capabilities with ever-increasing value
Role-specific processes and tools	Collaborative platform of integrated, web-based tools and practices
False, early precision in plans and requirements	Honest, evolving precision as uncertainties are resolved
Governance through measurement of artifact production and activity completion	Governance through measurement of incremental outcomes and progress/quality trends
Engineering discipline: track progress against static plans	Economic discipline: reduce uncertainties, manage variance, measure trends, adapt and steer

A. *Integration Testing Should Precede Unit Testing.*

In practice, integration and unit testing proceed in parallel. However, to accelerate the transformation to increased agility, it is best to demand that intermediate milestones include executable test cases of integrated functionality.

Integrating first to demonstrate the architecturally significant challenges first will resolve the big uncertainties earlier. It is human nature to address the easy things first to show early progress, but there is no economic leverage in doing so. By postponing the resolution of uncertainty, you decrease the probability of success.

Demanding an integration-first priority will help you increase the agility of the larger project team.

Project management. Lay out plans, resources, and measures that prioritize integration testing of key usage scenarios in multiple checkpoints as the primary steering mechanism.

Analysis. Analyze the business context or system context to define first the integrated behaviors, qualities, and usage scenarios that represent the holistic value and whose elaboration will reduce most of the project uncertainty.

Architecture. Elaborate the architecture of the solution and the evaluation criteria for incremental demonstration of the most important behaviors and attributes, such as changeability, performance, integrity, security, usability, and reliability.

Design/development. Develop units, services, and components that are always executable and testable, evolving from initial versions that permit execution within their usage context (that is, satisfy their interface) in a trivial way, and then progress toward more complete components that meet quality expectations across their entire operational spectrum.

Testing. Identify testing infrastructure, data sets, sequences, harnesses, drivers, and test cases that permit automated regression testing and integration testing to proceed without reliance on completely tested units and components.

B. *Agility and Control Must Be Complementary.*

In most conventional software engineering cultures, management governance *competes* with practitioner freedom. Here are two recurring observations from such cultures:

1. Where there is a perception of good governance, practitioners feel choked by repetitive manual reporting.
2. Where practitioner agility and morale are positive, management feels out of control with dynamically changing baselines.

To accelerate software delivery and achieve agility at scale, we need to integrate governance with agility so they are complementary objectives and not competitive forces. We must provide management with improved steering mechanisms such as progress and control measures, automated instrumentation, and real-time development analytics. We must also enable practitioners with more freedom to innovate through automation of measurement, traceability and change propagation, process enactment, reduced scrap and rework, and automated progress reporting.

The platform of know-how and automation must deliver this critical *quid pro quo*:

More freedom and less overhead for practitioners

← →

Better measurement and predictability for stakeholders

Practitioners need to maximize their productive time to create product or service capabilities that delight their users. Practitioners must demand that overhead tasks be minimized or automated. Management stakeholders, on the other hand, need more insightful measures and a feedback control loop to steer progress and quality with better predictability of business outcomes.

Improving measurement discipline is important because it correlates strongly with better business performance. The data in Table 2, compiled from more than 150,000 projects across hundreds of software organizations by Jones [28], illustrates how measurement discipline can promote more trustworthy communications among stakeholders. The value of that trust can only be quantified coarsely, but the impact is eye-opening.

Covey [29] coined the term "the speed of trust" to illuminate the necessary ingredient in reducing overhead and improving efficiency: trust. Trust is the secret ingredient necessary to achieve the *measurement-freedom quid pro quo*. Trust is not just a matter of ethics; it is a matter of competence: track record, transparency, and accountability.

To reconcile these competing points of view by establishing more trust between constituencies, developers need to embrace meaningful measurements, and management needs to enable and empower agility.

The following section describes how this crucial exchange between governors and practitioners (the "give-to-get" agreement of a *quid pro quo*) can enable economic governance in a large-scale enterprise context.

IV. Deploying Agile Know-how at Enterprise Scale

In the past 10 years, we have introduced agile techniques in hundreds of complex enterprise delivery contexts [6], [30]. Such enterprise organizations typically have these characteristics:

1. Diverse software supply chains across many business units, working in many geographical regions and governed by multi-layered reporting, that place extreme demands on collaboration and integration.
2. Complex organizational, technical, and economic structures that demand complex decision frameworks and complicate meaningful measures.
3. Blended technical platforms, processes, and tools for software development and delivery that require an incremental approach to change.

To manage these complexities, IBM's approach includes a strong measured improvement point of view, supported by adoption guidance for translating agile methods into incremental outcomes. There are two key elements to our approach. First, we define a framework for prioritizing agile practice areas. Second, we introduce a performance measurement framework that emphasizes and encourages an incremental, measured approach to agile practice adoption.

Table 2. How important is measurement?

Measurement practice	Strong	Weak
On-time projects	75%	45%
Late projects	20%	40%
Cancelled projects	5%	15%
Defect removal efficiency	95%	<85%
Resource estimates	Accurate	Optimistic
Client satisfaction	Higher	Lower
Staff morale	Higher	Lower

A. *A Software Practice Framework*

The key to adopting agility at scale is to prioritize the important practices and to segment them into a phased approach to improvement. We have seen several agile adoption efforts experience significant roadblocks when they are overly focused on specific techniques and tools, rather than on improving the practices that are most important to the organization. It is far too easy to place all the attention on the activity and lose track of the intended outcome and measured improvement that the implementation is intended to deliver.

We have found that five core practices constitute the key prerequisites for efficient adoption of agile approaches.

1. Iterative development
2. Two-level planning
3. Whole team thinking
4. Continuous integration and delivery
5. Test-driven development

Addressing these core practices early achieves an explicit and shared understanding among key constituents. If an organization is not experienced in iterative development techniques, the direct transition to an agile, continuous delivery approach is too much change to swallow at one time. Two-level planning and whole team thinking reflect the predominant project management shifts. The shift to outside-in assessment (test-driven development) and continuous integration ensure that uncertainties are resolved earlier, resulting in improved predictability of outcomes.

In support of the agile core practices, our framework identifies five additional areas of improvement: governance and compliance management, requirements management, change and release management, architecture management, and quality management. By examining other software engineering improvement frameworks such as IBM's Rational Unified Process (RUP) and CMMI, these five areas represent the critical areas of concern for most organizations, and offer a comprehensive framework for measured improvement. In practice, several detailed discussions and workshops may be necessary to identify the specific strengths and weaknesses of the organization, prioritize them, and set target improvement objectives. For example, some organizations may focus on areas of specific competitive advantage (such as speed of delivering new features), on areas of known shortfalls (such as improving software quality), or on alignment with other on-going transformations (such as a consistent approach to requirements management in concert with business analysts).

We need to define several dimensions of each practice:

- *Key concepts.* What are the key principles of the practice, and how does it relate to other practices?
- *Work products.* Which artifacts are critical for this practice, and how will they be maintained?
- *Tasks.* What are the most important activities and usage models for that practice?
- *Guidance.* What advice and heuristics are useful for the practice?
- *Automation.* How can tooling be applied to reduce overhead work and improve efficiency and quality?
- *Measurements.* What are the key measures for the practices, what is the current benchmark, and what are reasonable improvement targets?

This last dimension is a key discriminator of success. In prioritizing these practices, it is essential to define and agree on the key measured improvement targets for each practice across the organization. Based on this approach, the organization can create priorities around the areas for improvement. Practices are adopted incrementally based on a measured plan for improvement that is well understood across the organization.

B. Meaningful Performance Metrics

One principle of success for any organizational change is to adopt change incrementally based on measured feedback. Therefore, it is essential that an agile delivery adoption approach is supported by measures and instrumentation that provide insight into progress and quality improvements.

Measurement is an important and complicated challenge for any enterprise software delivery effort. Much has been written on this topic, with particular attention in recent years on cost control and efficiency in enterprise software delivery. Measurement know-how can be applied to the context of agile delivery rollout. In practice, we have found that the most critical first step is to understand what ideally *could* be measured, from what practically *can* be measured, from what effectively *is* measured. Too often we see agile delivery adoption programs that include sophisticated measuring schemes where the organization has little history, maturity, or context for those measures. The result is that such schemes become abused, mistrusted, and ignored.

No single set of measures for agile delivery adoption will work for all organizations. Rather, in adopting agile principles in a large-scale enterprise, we recommend clear, explicit goals that help to steer the organization toward its objectives. Table 3 shows a simple three-level measurement approach that we used with one large banking organization to baseline their goals and priorities for agile delivery adoption. The goal of this agile delivery adoption scheme was to address three basic questions.

TABLE 3. A FRAMEWORK FOR MEASURING AGILE DELIVERY ADOPTION

Business Measures	**IT Measures**	**Agile Measures**
Delivery cycles	Overhead	Practice adoption
Productivity and cost reduction	Change efficiency	Role adoption
Quality and customer satisfaction	Requirements churn	Work product adoption
Skills and reputation	Integration stability	Task adoption
Employee satisfaction	Test time	Process adoption

1. *Are we meeting the business objectives?* The original business objectives for the agile delivery program raised expectations with executive management that the enterprise software delivery organization would be faster-cheaper-better. Simple quantitative measures were specified to correlate the technical measures with the business outcomes expected.
2. *Are we seeing the benefits we expected?* Technical measures were identified as key performance indicators (KPIs) that quantified different perspectives of status and progress. The agile delivery adoption program was aligned with those KPIs to rationalize forecasted improvement plans and to estimate expected contributions.
3. *Are we truly agile?* Individuals and teams in the organization had their own view of what they expected from the agile delivery program. Continual assessment took place, through surveys and qualitative interview-based approaches, to understand how the agile delivery program was proceeding and to normalize perceptions with objective benchmarks.

The ability to obtain accurate, timely measures is critical to this simple approach to measured improvement. Our experience has been that the individual practitioners place greater emphasis on improvement if there are a small number of metrics derived directly from the model, code, and test base with clear interpretation. Table 4 shows a good starting point for specific measures.

When starting agile adoption, it is important that the business owners deliver more software, more quickly, and with better quality. Developers and engineers will embrace automated measurement of the engineering artifacts (models, code, and test bases). Automated instrumentation eliminates the overhead drudgery of metrics collection, progress reporting, change propagation, and traceability, and increases transparency and accountability. Likewise, when governance authorities are enabled with more *honest* measures that correlate better (and well) with dynamically changing progress and quality trends, they will encourage more change. Royce provides a more comprehensive treatment of measuring agility and architectural integrity in [22].

TABLE 4. SIMPLIFIED MEASUREMENTS FOR AGILE SOFTWARE DELIVERY

Measure	Business-Related	Team-Related
Cycle-time reduction	Time from initiation to delivery of first increment Time from initiation to project closure	Build/release cycle time Sprint velocity Blocking work items Requirements → tests Change costs over time
Quality	Production defects per 100 function points	Defect trends Change trends Integration trends
Continuous optimization	Process maturity level	Practice adoption Variance in cost to complete
Productivity	Function points per man-year	Sprint burndown chart Release burndown chart

Constant monitoring of the health of the agile delivery program results in both governance authorities and practitioners embracing the measurement with shared objectives. This can be used as the basis for early decision making, and provides visibility into the program for all of those involved.

The final section describes a specific framework through which the principles of economic governance and measured improvement can be applied to enterprise-scale software delivery projects.

V. A METHOD FOR DISCIPLINED AGILE DELIVERY

People new to modern best practices, and even many with agile experience, often ask fundamental questions: How does architecture fit into agile delivery? When does it make sense to write requirements specifications? What level of detail is necessary? When should you enhance your whole team testing efforts with an independent test group? When shouldn't you? How do you work successfully with outside teams when they are using traditional methods?

To address these concerns, it is essential to have a framework that supports a coherent, end-to-end strategy for improvement; this includes a solid understanding of how agile solution delivery succeeds in practice.

Our experience with large enterprises has led to the development of the Disciplined Agile Delivery (DAD) process framework [31]. The DAD process framework has several important characteristics. These characteristics are presented in priority order from the point of view of enabling the scaling of your agile process.

People first. We foster the strategy of cross-functional teams made up of cross-functional people. There should be no formal hierarchy within the team, and team members are encouraged to be cross-functional in their skill set and perform work related to disciplines other than their specialty. The increased understanding that team members gain beyond their primary discipline results in more effective use of resources and reduced reliance on formal documentation and signoffs. This in turn enables scaling through easier communication and collaboration.

Explicit scaling support. The DAD process framework is an important part of IBM's strategy, which explicitly promotes that there is more to scaling than team size and there are multiple scaling factors a team may need to address. These scaling factors are team size, geographic distribution, regulatory compliance, domain complexity, technical complexity, organizational distribution, organizational complexity, and enterprise discipline. Each team will be in a unique situation and will need to tailor its strategy accordingly. For example, a collocated team of seven in a regulatory environment will work differently than a team of 40 spread out across several locations in a nonregulatory environment. Each of the eight scaling factors will motivate teams to tailor the foundation practices.

Goal-driven. The framework is goal-driven, as summarized in Table 5. For each goal, we describe several issues that must be considered. We show several options for addressing each issue and the trade-offs for each option. Suggestions are provided for what we've found to work best; there is no prescription. Context matters and the starting points in our framework require some elaboration into each specific situation. For example, consider the goal of identifying the initial requirements for your project. There are several issues that need thoughtful consideration when tailoring your process: What level of detail (detailed specification, lightweight specification, a list of goals, none) are you going to capture? What types of modeling (domain, process, user interface, nonfunctional) will you perform? For each type, how will you explore that issue?

TABLE 5. SOME GOALS ADDRESSED THROUGHOUT A DAD PROJECT

Inception Phase Goals	Construction Phase Goals
* Form initial team * Develop common project vision * Align with enterprise direction * Explore initial scope * Identify initial technical strategy * Develop initial release plan * Form work environment * Secure funding * Identify risks	* Produce a consumable solution * Address changing needs * Move to deployable release * Improve quality * Prove architecture early

For example, usage modeling can be addressed via user stories, use cases, usage scenarios, feature statements, and more. What elicitation strategies (formal, informal, interviews) will you use? What change management strategy (formal, product backlog, work item list, work item pool) will you adopt? How will you capture (through technical stories, acceptance criteria, an explicit list, or not at all) the pertinent nonfunctional requirements? Although identifying initial requirements is a straightforward goal, satisfying this goal in a way that reflects the actual situation often requires more than simply creating a stack of index cards.

This goal-driven approach avoids the tailoring challenges associated with more prescriptive methods such as Scrum or UP. Where Scrum starts with a small base from which you need to add process, or UP starts with a large base from which you remove process, the DAD approach starts in the middle ground and provides sensible tailoring advice. Your tailoring choices will be driven both by the scaling factors faced by a team as well as the skills and culture of that team.

Enterprise awareness. Agile delivery teams generally live within a larger context. There are often systems currently in production, and minimally your solution shouldn't impact them, although your solution could leverage existing functionality and data available in production. There may be teams working in parallel to your team; you might take advantage of some of what they're doing, and vice versa. Your organization may be working toward a common vision which your team should contribute to. There will be a governance strategy in place, although it may not be obvious to you, that enhances what your team is doing. Effective teams work in a way that reflects these realities.

Risk and value driven. Our process framework adopts a risk/value lifecycle, which is a lightweight version of the strategy promoted by UP. Teams strive to address common project risks, such as coming to stakeholder consensus around the vision and proving the architecture, early in the lifecycle. Explicit checkpoints are defined for continued project viability and for assessing whether sufficient functionality has been produced and the solution is production ready. It is also value-driven, a strategy that reduces delivery risk, in that teams produce potentially consumable solutions on a regular basis, which can be deployed with positive results ahead of project schedule, if necessary.

Delivery-focused. The lifecycle context ranges from initiation through construction to releasing your solution into production. It also shows some pre-initiation portfolio management activities as well as post-delivery production activities. This expands on first-generation agile methods, which typically focus on the construction aspects of the lifecycle. DAD also supports a lean, or advanced, lifecycle, to which many agile teams evolve after applying process improvements over time.

IT solution focused. IT professionals do far more than just develop software. While software is clearly important, in addressing the needs of our stakeholders we often provide new or upgraded hardware, change the business/operational processes that stakeholders follow, and even help change the organizational structure in which our stakeholders work.

Agile. Our process framework adheres to, and enhances, the values and principles of the Agile Manifesto [1]. Teams following either iterative or agile processes have demonstrated higher quality, provide greater return on investment, provide greater stakeholder satisfaction, and deliver more quickly than teams using either a traditional/waterfall approach or an *ad hoc* (no defined process) approach.

Hybrid. The approach is hybrid in that it adopts and tailors proven strategies from methods such as Scrum, Extreme Programming (XP), Agile Modeling (AM), Unified Process (UP), Lean Software Development, Kanban, and Agile Data (AD). Consequently, DAD is a second-generation agile process framework that leverages what has come before it.

Learning-oriented. There are three key aspects of an effective learning environment. The first is domain learning: How are you exploring and identifying your stakeholder needs? Perhaps more importantly, how are you helping your team do so? The second aspect is improving your process at the individual, team, and enterprise levels. The third aspect is technical learning, understanding how to work effectively with the tools and technologies being used to craft the solution.

VI. Conclusions

Many organizations are now moving from small-scale agile development success toward larger-scale agile delivery excellence. Our transformational approach led us to focus on three key areas: economic governance, measured improvement, and disciplined agile delivery.

Economic governance is not a new idea. The principles of the spiral model [3], iterative development [4], and risk management [32], and the foundations of modern agile methods such as test-driven development all exemplify a process spirit that emphasizes continuous and early integration. To translate these principles into better outcomes, project teams need to plan their activities and early releases to drive integration testing priorities before unit testing.

In the most standout software successes we have observed, where software product releases are straightforward to maintain over a long period, teams prioritize continuous integration testing of the forest over detailed unit testing of the trees. The two strongest, industrial-strength examples of integration-first results are well-documented [24, Appendix D], [33]. This integration-first spirit has a natural parallel in managing systems and software development teams: Optimizing collaborative teamwork is more important than optimizing individual productivity.

Measurable improvements in defect profiles and reduced cycle times for change are the key agile attributes that differentiate the winners from the also-rans in improving software economics. In this context, measured improvement means measuring the progressions and digressions in continuous integration milestones. These are the important windows into improved economic governance that improve collaboration and resolve uncertainties in their process and product earlier.

Agility means quick to react. Change speed must be an asset, not an anchor. Agility is best achieved by demanding that integration testing be performed earlier and continuously.

Agility is best measured by quantifying the costs of change over time. Teams that have improved their turnaround time for software changes from a few weeks to a few days have clearly become more agile. Instead of being mired in late scrap and rework, and consumed playing defense, they can go on the offensive by innovating more, adding more features or more quality, improving performance, or delivering earlier.

The topics of discipline and control in agile software delivery can be sensitive for many organizations. Too many process owners advocate extreme positions on both ends of the process rigor spectrum. The *traditional* extreme uses overly comprehensive process descriptions: the IBM RUP, Team Software Process (TSP), and CMMI, for example. The challenge with RUP was usually that teams didn't scale it down appropriately, which often resulted in an excess of artifacts and too much overhead. The *progressive* extreme uses an overly lightweight process description, with Scrum being the exemplar. Many Scrum teams could not scale up appropriately, resulting in significant effort reinventing techniques to address the myriad issues beyond core software development which Scrum doesn't cover. A lot of waste could have been avoided if there had been an option between these two extremes. Disciplined Agile Delivery is a more balanced option.

There are probably more books on agile methods than successful projects with well-documented agile results. Writing a book on agility or project management, where the decision-making process is laid out in the abstract, is easy, compared to managing a real project where you must steer through a minefield of uncertainties and the consequences of decisions under game conditions are very real. We should glorify the practice leaders in our industry and the people who publish measured improvement case studies. They are critical to accelerating the innovation delivered in software. Practice leaders resolve uncertainties and steer projects from the abstract to the real. The demand for such practice leadership is much higher than the demand for thought leadership.

REFERENCES

1. The Agile Manifesto, http://agilemanifesto.org/
2. P. Naur and B. Randell (eds.), "Software engineering: report of a conference sponsored by the NATO Science Committee," Garmisch, Germany, 7-11 October 1968. http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF
3. B. Randell and J.N. Buxton (eds.), "Software engineering techniques: report of a conference sponsored by the NATO Science Committee," Rome, Italy, 27-31 October 1969. http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF
4. B. Boehm, "A spiral model of software development and enhancement," ACM Sigsoft Softw Eng Notes, ACM, vol. 11, no. 4, pp. 14-24, August 1988.
5. P. Kruchten, Rational Unified Process. Addison-Wesley, 2002.
6. CMMi for development, version 1.3, CMU/SEI-2010-TR-033, November 2010.
7. H. Glazer, "CMMi and agile: why not embrace both?," SEI Technical Report, CMU/SEI-2008-TN-003, 2009.
8. S.A. Ambler, "The agile scaling model (ASM): adapting agile methods for complex environments," IBM Developer Works, December 2009.
9. P. Kruchten, "Contextualizing agile software development," J Softw Maint Evol-R, 2011.
10. D. Leffingwell, Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise. Pearson Education, Inc., 2011.
11. E. Woodward, A Guide to Distributed Scrum. IBM Press, Pearson Publishing, 2010.
12. C. Larman and B. Vodde, Scaling Lean and Agile Development: Thinking and Organizational Tools for Large-Scale Scrum. Addison-Wesley, 2009.
13. A.W. Brown, "A case study in agility-at-scale delivery," Proc 12th Int Conf on Agile Software Development, XP2011, Springer Verlag, May 2011.
14. G Benefield, "Rolling out agile in a large enterprise," Proc IEEE Comput Soc HICSS08, 2008.
15. I. Oshri, J. Kotlarsky, J.W. Rottman, and L.P. Willcocks, "Global sourcing: recent trends and issues," Inform Technol People, vol. 22, no. 3, pp. 192-200.
16. D. Leffingwell. Agile Software Requirements, Addison-Wesley, 2010.
17. A. Highsmith. Agile Project Management: Creating Innovative Products. Addison-Wesley, 2009.
18. D. West, "Water-scrum-fall is the reality for most organizations," Forrester Analyst Report, July 26, 2011.
19. J. Gorman, "The Wagile software development lifecycle," 2008. http://www.codemanship.co.uk/parlezuml/blog/?postid=708
20. J. Sutherland, "Ten year agile retrospective: how can agile improve in the next ten years?" 2011. http://msdn.microsoft.com/en-us/library/hh350860.aspx
21. W.E. Royce, K. Bittner, and M. Perrow. The Economics of Software Development. Reading, Massachusetts: Addison-Wesley, 2009.
22. W. Royce, "Measuring agility and architectural integrity," Int J Softw Info, vol. 5, no. 3, 2011, ISCAS.
23. W.W. Royce, "Managing the development of large software systems," Proc. IEEE Wescon, pp. 1-9, August 1970.
24. W. Royce, Software Project Management, A Unified Approach. Addison-Wesley, 1998.
25. M. Kennaley, SDLC 3.0, Beyond a Tacit Understanding of Agile. Fourth Medium Press, 2010.
26. S.A. Ambler, "IT governance and project management survey results," July 2009. http://www.ambysoft.com/surveys/stateOfITUnion200907.html
27. S.A. Ambler, "How agile are you? 2010 survey results," 2010. http://www.ambysoft.com/surveys/howAgileAreYou2010.html
28. C. Jones, Software Engineering Best Practices. McGraw Hill, 2010.
29. S.M.R. Covey, The Speed of Trust: The One Thing That Changes Everything. Free Press, 2006.
30. A.W. Brown, Global Software Delivery: Bringing Efficiency and Agility to the Enterprise. Addison-Wesley, 2012.
31. S.W. Ambler and M.J. Lines, Disciplined Agile Delivery: A Practitioner's Guide to Agile Solution Delivery in the Enterprise. IBM Press, 2012.
32. B. Boehm, Software Risk Management. IEEE Computer Society Press, 1989.
33. C. Berggren, J. Jarkvik, and J. Soderlund, "Lagomizing, organic integration, and systems emergency wards: innovative practices in managing complex systems development projects," Project Manage J, vol. 39 (supplement), 2008, pp. S111-S122.