Applying agile and lean principles to the governance of software and systems development

Applying lean thinking to the governance of software development

As more and more project teams adopt agile software development methods, issues arise with traditional approaches to IT governance. Such approaches—which include Control Objectives for Information and Related Technology (COBIT) and the Project Management Institute (PMI) Organizational Project Management Maturity Model (OPM-3)—are often too heavy in practice for development shops seeking to be more agile or lean.

Effective governance for lean development isn't about command and control. Instead, the focus is on enabling the right behaviors and practices through collaborative and supportive techniques. It is far more effective to motivate people to do the right thing than it is to force them to do so.

A lean approach to development governance weaves the philosophies of lean and agile software development into traditional IT governance to form a lightweight, collaboration-based framework that reflects the realities of modern IT organizations.

This paper begins with an overview of the principles of effective IT governance, discussing how they pertain to the more specific subset of development governance. It follows with an overview of the principles of lean software development. The heart of the paper is a description of eighteen practices that define a lean approach to governing software development projects within your IT organization. These practices show how to embed governance into tools, processes and development guidance to make it as easy as possible for people to keep their software and systems delivery projects on track. They bring the lean thinking that has revolutionized supply chain management into the governance of software and systems development.

It's possible to loosen the reins on development teams without losing control.

The good news is that you're likely following some of these practices already. The bad news is that it'll likely take your organization several years to adopt every practice, and there's no set order in which to implement them. However, the practices are synergistic and can be adopted incrementally. Each added practice helps strengthen your ability to effectively implement subsequent practices.

Defining governance

IT and systems engineering governance establishes chains of responsibility, authority and communication in support of the overall enterprise's goals and strategy. It also establishes measurements, policies, standards and control mechanisms to enable people to carry out their roles and responsibilities effectively.[1] You do this by balancing risk versus return on investment (ROI), setting in place effective processes and practices, defining the direction and goals for the department, and defining the roles that people play with and within the department.

Governance and management are two different things: Governance looks at an organization from the outside, treating it as a system that needs to have the appropriate structure and processes in place to provide a stream of value. Management, on the other hand, is inside the organization and ensures that the structure and processes are implemented effectively.

Development governance is an important subset of IT and system engineering governance, the scope of which covers the steering of software and system development projects. Based on considerable experience with agile development teams, IBM believes organizations that have adopted COBIT or OPM-3 (or similar traditional frameworks) can benefit by loosening the reins on their approach to governing development projects—without sacrificing essential controls and audit trails.

To help you improve your governance framework, IBM has developed a palette of practices that support lean development.

Although many of these practices also apply to IT and systems engineering governance in general, the focus of this paper is on governing agile and lean development projects.

Principles of effective IT governance

In The IBM Rational Edge[2], Murray Cantor and John D. Sanders describe seven principles to guide IT governance efforts. Because development governance is a subset of IT governance, these principles are directly applicable to a discussion of lean development governance. They address process, artifact lifecycle, risk, suitability, behavior, deployment and automation.

Process

Governance is a process that is applied to the processes that need to be governed. Policies and standards are applied to development processes; decision rights are enforced within the processes; and the processes are measured and controlled.

Artifact lifecycle

The lifecycles of artifacts produced by the governed processes guide the governance solution. Part of governing software development is understanding the artifacts (such as executable tests and working software) produced by project teams, and then monitoring how these artifacts evolve throughout the develop¬ment effort.

Risk

Measures and controls must be adjusted according to the level of risk. When risk is low you can focus on measuring the activities of the team. When risk is high you need to focus on helping the team mitigate the risk(s) it faces.

Suitability

The needs of the organization determine how the level and style of governance are tailored. For example, development teams building life critical software need a tighter governance approach than teams building an informational Web site.

Behavior

The governance solution drives the organizational behavior. From a development governance point of view, the challenge is to create an environment that results in the development organization performing in such a way that business goals are met. In many ways the maxims "You ship the organization" and "You get what you measure" are primary motivators for organizations to build an effective development governance program.

Deployment

The governance solution must be implemented incrementally. You cannot implement all of the practices described in this paper at once. You must first pick and choose the ones that address your immediate pains before adopting others.

Automation

Technology makes the governance solution empowering and unobtrusive. The more you embed, or automate, your development governance practices through automation and culture, the more likely they are to be followed.

Principles of lean software development

In Implementing Lean Software Development [3], Mary and Tom Poppendieck show how the seven principles of lean manufacturing can be applied to optimize the whole IT value stream. These principles offer practical, measurable ways to transform software delivery processes.

Eliminate waste

Lean thinking advocates regard any activity that does not directly add value to the finished product as waste. The three biggest sources of waste in software development are the addition of extra features, churn and crossing organizational boundaries. Crossing organizational boundaries can increase costs by 25 percent or more by creating buffers that slow response time and interfere with communication. It is critical that development teams be allowed to organize and operate in a manner that reflects the work they're trying to accomplish—rather than the functional roles of team members.

Build in quality

The Poppendiecks make a simple observation: if you routinely find problems with your verification process, then your process must be defective. When you regularly find that your developers are doing things that you don't want them to

do—or are not doing what they should be doing—then your approach to governance must be at fault. It's important not to make governance yet another set of activities layered on top of your software process. Instead the strategy should be to embed governance into your processes, making it as easy as possible for developers to do the right thing.

Create knowledge

Planning is useful, but learning is essential. You want to promote strategies, such as iterative development, that help teams discover what stakeholders really want and act on that knowledge. It's also important to have a body of reusable standards and guidelines that people can easily modify to meet specific project needs. In addition, consistent and timely feedback is important, both within the team and at the program level, through continuous monitoring of simple and relevant metrics.

Defer commitment

It's not necessary to start software development by defining a complete specification. You can support the business effectively through flexible architectures that are change tolerant and by scheduling irreversible decisions to the last possible moment. Frequently, deferring commitment requires the ability to closely couple end-to-end business scenarios to capabilities developed in multiple applications by multiple projects.

Deliver quickly

It is possible to deliver high-quality systems quickly. By limiting the work of a team to its capacity, you can establish a reliable and repeatable flow of work. An effective governance strategy doesn't demand teams do more than they are capable of, but instead asks them to self-organize and determine what they can accomplish. At an organizational level, it's important to enable programs to deliver business value at a pace defined by the fastest-moving projects, rather than at the speed of the slowest project.

Respect people

The Poppendiecks also observe that sustainable advantage is gained from engaged, thinking people. The implication is that you need a human resources strategy that focuses on enabling IT teams—not on controlling them.

Optimize the whole

If you want to govern your development efforts effectively, you must look at the bigger picture. You need to understand the high-level business processes that individual projects support—processes that often cross multiple systems. You need to manage programs of interrelated systems so you can deliver a complete product to your stakeholders. Measurements should address how well you're delivering business value, because that is the raison d'être of your IT department.

Categories of development governance

Traditional governance often uses command-and-control strategies. These strategies focus on managing and directing development project teams explicitly, using gateways and triggers that attempt to enforce rules and catch violations. Although valid and effective in some situations, this approach can be like herding cats for many organizations. Much work is put into establishing the governance framework and managing the governance effort, but very little is achieved in practice.

In contrast, lean governance focuses on collaborative strategies that strive to enable and motivate team members implicitly. For example, the traditional approach to coding guidelines would be to create them and then enforce their usage through formal inspections and post hoc correction of errant code. The lean approach would be to write the guidelines collaboratively with your programmers, explain why it's important for everyone to adopt the guidelines, and then provide tooling and support to make it as easy as possible for developers to continuously code within those guidelines.

Figure 1 categorizes and illustrates the relationships of the practices for lean governance. It shows how they align to the six major categories of IT governance: mission and principles, organization, processes, measures, roles and responsibilities, and policies and standards.

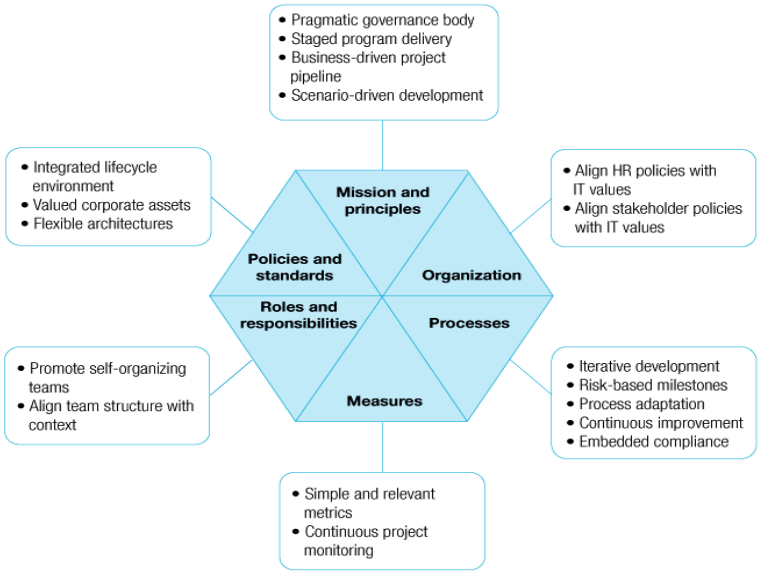

Figure 1: The eighteen practices of lean software development governance are aligned to six categories of governance.

Practices for mission and principles

IBM identifies four lean practices[4] to guide the mission and principles category of governance.

Pragmatic governance body

A governance program does not run itself; a group of people called a governance body runs it. The manner in which the governance body organizes and conducts itself is a key determinant of the overall effectiveness of the governance program.

To support lean development, a pragmatic governance body respects people by focusing on enabling IT professionals first, and on controlling and managing them second. It does this by creating an environment where people can be effective in practice and not just in management theory. Such an environment promotes situation-specific strategies, procedures and practices; provides teams with access to the resources they need, including ready access to business stakeholders; and provides guidance, support and mentoring to teams that have deviated from expected norms.

Under the guidance of a pragmatic governance body, IT teams will be much more likely to conform to the governance program because it's easy for them to do so. People will actually step up and make processes and polices come alive to help the organization reach its goals. The alternative is an environment where teams will do whatever is necessary to comply with the command-and-control governance structure. After any review, they'll simply return to managing the project the way they see fit, resulting in two sets of books—the real one and the one presented outside of the team.

An effective approach for lean development is to create a small central team, often referred to as a governance competency center, which is extended with part-time members from the governed IT organizations and the appropriate business units. Making key representatives of the governed organizations part of the governance body avoids a "us versus them" mentality and helps ensure that governance guidance is relevant and actionable. Picking the right people for the governance body is absolutely crucial. Very often the people who jump at the chance to volunteer for a governance body are the people you least want governing.

Staged program delivery

Staged program delivery enables you to optimize the whole program while still completing projects quickly. Programs, which are collections of related projects, should be rolled out incrementally over time. Instead of holding back a release to wait for a subproject, each subproject must sign up for a predetermined release date. If the subproject misses the date, it skips to the next release, minimizing the impact to the customers of the program. Think of a train schedule. If the project misses the release train, it has to wait for the next one. Granted, because of dependencies between projects, sometimes one project missing the release train causes several to do so.

As you see in Figure 2, usage of a control project enables coordinated execution of the program while providing flexibility in execution of individual projects within the control project. Iterations play a fundamental role here, because they provide stable anchoring points to allow projects to drive and validate meaningful cross-project integration.

There are several benefits to staged program delivery. First, by grouping projects according to business objectives, and managing them as a program, you can more effectively deliver on major business objectives. Second, using a control project provides well-defined governance milestones for the program that focus on risk and variance reduction as well as value creation. Third, dividing a potentially large program delivers value incrementally around business subgoals. And finally, semi-independent execution of projects enables more efficient development because each project has as much tactical flexibility as possible to support higher productivity.

It is a good strategy to manage programs with loosely coupled individual projects, by a control project run according to four phases: inception, elaboration, construction and transition. Ideally the individual projects should take an evolutionary approach, using agile processes such as Disciplined Agile Delivery (DAD), Scrum or OpenUP.

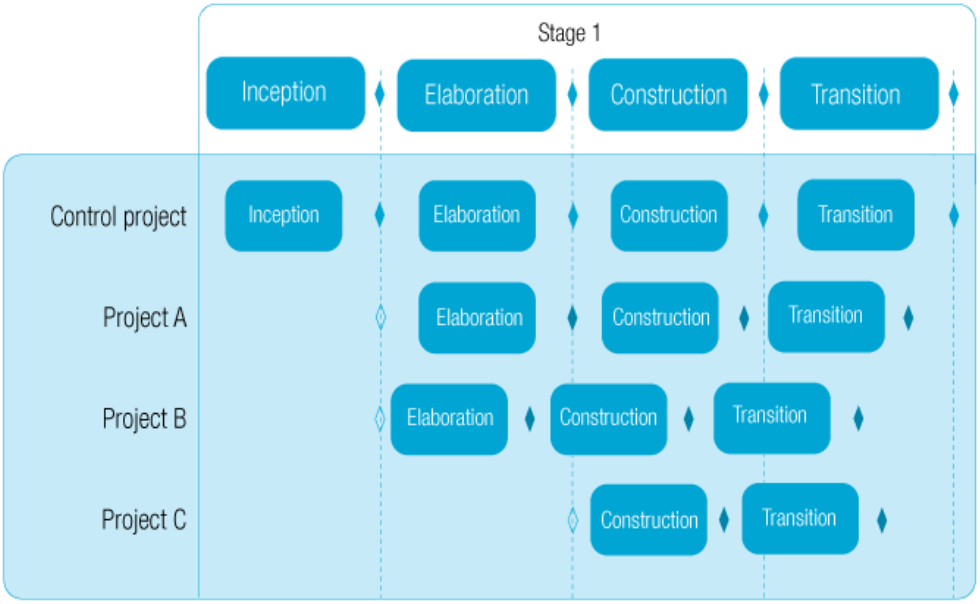

Figure 2: As illustrated by Bittner and Spence in Managing Iterative Software Development Projects, risk reduction and value creation can be effectively addressed by managing a program stage through a control project.

Business-driven project pipeline

The demand for IT projects always exceeds available resources. A business-driven project pipeline maximizes the business value of development investments by enabling organizations to prioritize and optimize projects in alignment with business goals and objectives. This alignment can be accomplished with scorecards and other portfolio management strategies that help you assess each project against a set of parameters you define to measure business value. Lean thinking encourages organizations to focus on no more than five parameters and use scoring as an aid in a business discussion about prioritization rather than as a strict measure.

Scenario-driven development

Scenario-driven development provides the business context in which to drive effective software development projects. The whole cannot be defined without understanding the parts, and the parts cannot be defined in detail without understanding the whole. If you don't know how the parts impact the overall solution, you can get bogged down in building components that don't fit together. To understand the big picture of the business, you can identify usage scenarios at both the enterprise and project levels using approaches such as use-case flow down[7] or green threads.[8] These techniques help each project team see how its part fits into the whole. They also improve collaboration among and between teams and serve as consistent control mechanisms by focusing developer attention on system integrations.

Practices for organization

Two lean practices guide definition of the appropriate organizational climate to support agile development.[9] Applying these practices can help you better leverage people as a critical resource.

Align HR policies with IT values

Hiring, retaining and promoting technical staff requires different strategies than those for non-IT staff. To reward desired behaviors, you need to ensure that incentives are appropriate for the mind-set of your technical staff. For example, many IT professionals want to expand their technical skills so they can work on more challenging projects. Yet most are not interested in managing large teams of people. You will lose qualified people if the only senior roles in your IT organization are management positions. Effective human resources (HR) policies can help you increase your pool of resources and retain skilled staff.

Align stakeholder policies with IT values

You can easily derail software development projects with ill-fitting business practices. For example, many organizations will insist on an "accurate" cost estimate at the beginning of a software development project. This proves unrealistic in practice because requirements evolve over the life of a project[10] and because there are so many uncertainties in a project, which leads to a variance in your estimate.

Besides being unrealistic, stakeholder demand for an accurate estimate up front motivates development teams to adopt risky practices such as detailed requirements definitions early in the project.[11] Although it is expected that the business will put cost and schedule constraints on software development teams, the way in which it does so must reflect the realities of the develop-ment lifecycle and must be flexible enough to allow development teams to remain effective.[12]

In short, your business stakeholders must have realistic policies for engaging IT, including how projects are funded, how requirements are documented, and the level and kind of involvement stakeholders will have with software development project teams.

Lean thinking encourages business stakeholders to be active participants on development teams. IT professionals should be responsible for educating stakeholders in the fundamentals of modern software development so they understand the options available to them and the implications of their decisions. Conversely, IT professionals should learn the fundamentals of business value management so they can provide the earliest possible warning when project costs are in danger of exceeding the expected business value. The benefits of these practices include increased probability of project success and improved software economics due to improved decision making.

Business stakeholders must play an active role on development project teams.

Practices for development processes

Five lean practices promote strategies for running a project efficiently and effectively.[13]

Iterative development

Using an iterative development approach, a project is organized into a sequence of short iterations (referred to as "sprints" in Scrum[14]). Each iteration is time boxed and has a well-defined set of objectives. The aim of each iteration is com-plete, defect-free, working code.

During each iteration, you build on the work of previous iterations to evolve the requirements, analysis, design, implementation and test assets until the final product is complete (see Figure 3). There are several key benefits to this approach. First, the length of each iteration is fixed, not the scope. If the estimation of effort is off due to unforeseen events or difficulties, it is the scope that must give and be reassessed in the next iteration. Second, the definitions of "final product" and "complete" may evolve significantly from their original conceptions during project execution when the business stakeholders and implementers agree that doing so better satisfies business objectives. Each iteration of the project will increase the shared understanding of requirements. Third, time boxing forces fast decision making and a crisp focus on what matters most—the regular delivery of working software. Note that delivery isn't always into production. Sometimes it's just into a test environment. The regular delivery of working software increases the number of feedback opportunities because stakeholders can easily see whether IT understands what they've asked for. It also enables fact-based governance because working software is a concrete representation of what a team has accomplished; whereas secondary artifacts such as plans or specifications are merely promises that the team may deliver at some point.

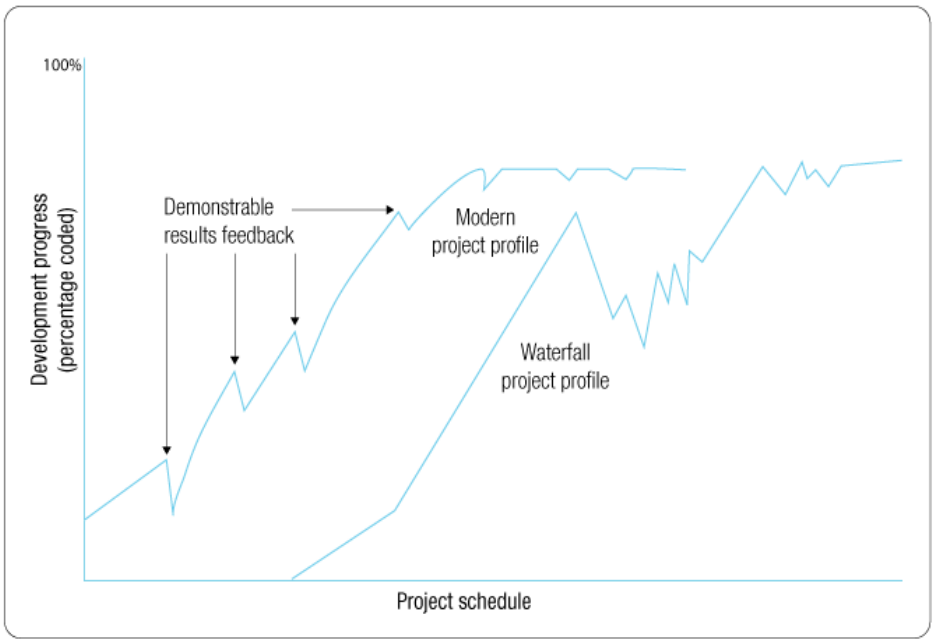

Figure 3: Based on frequent demonstrations and stakeholder feedback, the small course corrections early in the lifecycle of an iterative ("modern") project lead to more rapid success, compared to a "waterfall" project.

Most important, an iterative approach increases your ability to build systems that meet the changing needs of your stakeholders. IBM recommends four-week-long iterations by default.[15] As you recognize that some other length for iterations is more appropriate for the context of a project, change the iteration length to reflect that context. You should strive for the shortest iteration length possible for your environment because the longer the iteration length the greater the risk of allowing needless bureaucracy to creep into your software delivery processes. There should be no time between iterations: all activities—from evaluating progress through planning and executing the next iteration—are an integral part of an iteration, not a superimposed external process.

Risk-based milestones

Iterative development is most effective when you combine it with a deliberate balance of early risk reduction and early value creation through risk-based milestones. This means that, as you prioritize the work for each iteration, you choose to develop those features that represent the biggest business, organizational, programmatic and technical risks while delivering the most value. Because these two objectives—greatest risk and greatest value—are not usually aligned, the risk-based milestones approach of Disciplined Agile Delivery (DAD) or Unified Process (UP) forces a deliberate choice between maximizing early value creation and early risk reduction. Both are fundamental for project success (see Figure 4), so it is important to have the right control points in place.

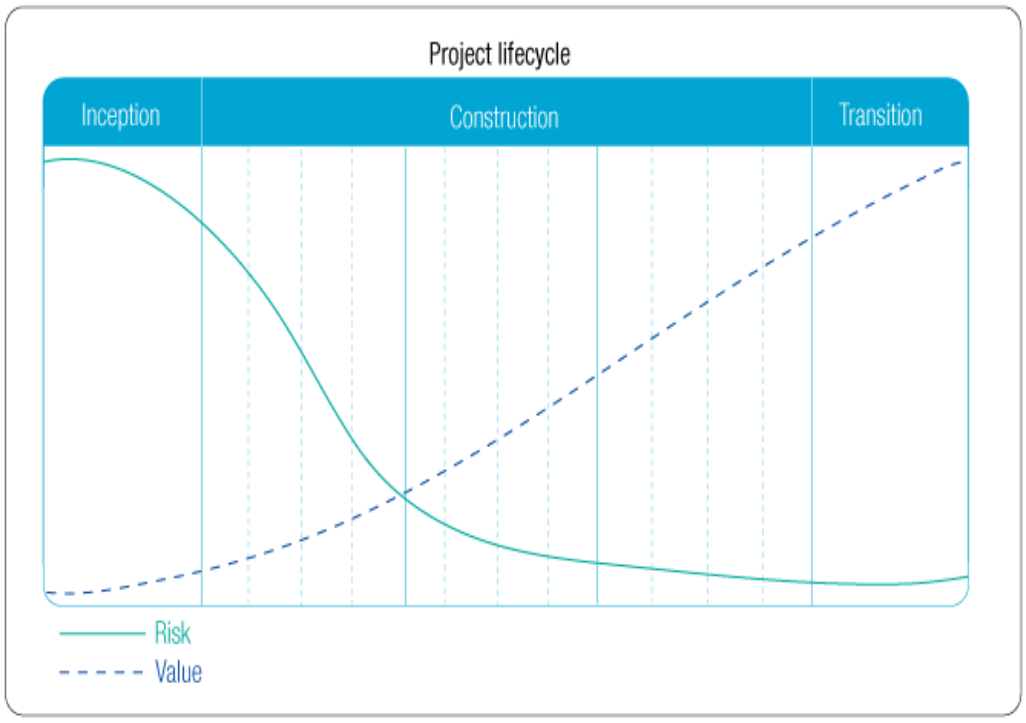

Figure 4: Risk reduction (teal curve) and value (dashed blue curve) during the project lifecycle

Throughout a Disciplined Agile Delivery (DAD) project there are light-weight management milestones requiring project deliverables to be assessed for risk reduction and value creation. For example, at the end of the Inception phase you should verify that a shared stakeholder consensus around vision and plan for the current release has been reached. Early in the Construction phase the team needs to demonstrate that it has an executable architecture, with a few selected scenarios that can be executed, and with a risk list that reflects the mitigation of many key technical and other risks. This risk reduction needs to be balanced with the value of the running code to show concrete evidence that the team has made actual progress. Note that Rational Unified Process (RUP) takes a slightly different approach with an explicit phase called Elaboration which focuses on proving the architecture with working code – DAD adopted the strategy of proving the architecture with working code from RUP but combined its Elaboration and Construction phases into one. IBM recommends adopting a risk-value lifecycle which includes explicit milestones.[16] You can adopt similar strategies even if you are following other agile processes such as Scrum or Extreme Programming (XP).

Risk-based milestones provide greater stakeholder insight and control and promote early value creation. They also help reduce the chance of project failure and can improve productivity by driving out technical risks early, helping to thereby reduce overall project risk.

Process adaptation

Because all projects are not created equal, it is critical to adapt the development process to the needs of the project. A team developing a Web site will work differently than a team developing a data warehouse. A team of 5 people will work differently than a team of 50. A team developing a life-critical system will work differently than a team building a business application. It is not a question of more process being better or less process being better. Rather, the amount of ceremony, precision and control present in a project must be tailored to a variety of factors. The Agile Scaling Model (ASM) [26] defines a contextual framework for understanding the range of situations which IT delivery teams face, agile or otherwise, calling out eight scaling factors: geographic distribution, team size, regulatory compliance, domain complexity, technical complexity, organizational complexity, organization distribution, and enterprise discipline.

We have two critical observations about tailoring your process to meet your needs. First, a flexible, practices-based process framework allows you to maximize reuse across lines of business (LOB) and projects while enabling flexibility so they can adapt the process to their needs. For example, a given project cares about eight specific practices right now because they address the challenges faced by that team. Another team may focus on six practices, with four in common with the other project team, because they face a different situation. The IBM practices library is a good resource [27]. Second, a project team should also adapt process ceremony to lifecycle phase. The beginning of a project is typically accompanied by considerable uncertainty, and you want to encourage a lot of creativity to develop an application that addresses the business needs. More process typically reduces creativity, so you should use less process at the beginning of a project when uncertainty is an everyday factor. On the other hand, late in the project you often want to introduce more control, such as feature freeze or change-control boards, to remove unpredictability and risks associated with the late introduction of defects.

Continuous improvement

Lean-thinking organizations strive to continually improve processes. IBM recommends performing an assessment, such as a retrospective,[17] at the end of each iteration and at project end to capture lessons learned, and leverage that knowledge to improve the process.

The fundamental concept behind continuous improvement is simple: improve the way you work whenever the opportunity presents itself. The old recommendation “You should learn something new every day” is a good one. Furthermore, this practice goes one step further and recommends that you act on what you learn and increase your overall effectiveness.

There are several ways that you can identify potential improvements to your software process during the execution of

a software project. For example, you could include informal improvement sessions or staff suggestion boxes or Web sites. Or you could encourage personal reflection and provide teams with an editable process (perhaps via a wiki). An effective strategy is to schedule two hours at the end of each iteration for an informal retrospective. The practice of continuous improvement enables you to learn as you go and gives the team clear control over its destiny.

Embedded compliance

The easier compliance is to achieve, the greater the chance IT professionals will actually comply. Therefore, compliance to regulations, corporate policies and guidance should be automated wherever possible. When automation isn't an option, compliance-related tasks should be part of your culture and daily activities, rather than a set of tasks done as a separate and late-in-the-game effort. Compliance becomes part of your culture when people understand why it's important and what the underlying principles are behind it. If compliance requires significant amounts of extra work, particularly when that work is perceived to be onerous or arbitrary by the people doing it, then chances are greater that your compliance effort will be subverted by development teams.

The benefits of embedding compliance include reducing your overall compliance costs, reducing pushback from development teams, and achieving higher levels of compliance than with traditional approaches. IBM recommends that you define a minimal solution that can be integrated into your process and tools based on the intent of the appropriate regulations. A key consideration is assigning the right people to interpret the regulations and create guidelines for development teams to follow. If you assign bureaucrats to this effort, you will end up with a bureaucratic solution.

Practices for measurement

Within the measures category of development governance, IBM recommends two lean practices that foster informed decision making with supporting targets and incentives.[18]

Simple and relevant metrics

Simple and relevant metrics provide the information necessary to understand how you are doing so you can take corrective actions as needed. Sadly, most organizations either use no metrics at all—meaning they're effectively flying blind—or they overdo it. They collect so many measurements that they drown in data.

Effective metrics are relevant because they are commonly used throughout your organization, and they provide the information managers and executives need to take timely, appropriate action. When done right, simple and relevant metrics enable fact-based governance. When such metrics are automated, they enable painless governance. And when they are used to explore critical trends, they enable proactive governance.

It's best to start with metrics that explore the value being delivered by teams, the quality being delivered by the project and the cost being expended. Not only are these metrics useful for determining the current status of your projects, but they're also useful for determining how far a team has deviated from initial expectations. Expectations are set at the business case level for time, spending and use of resources, and may be updated periodically throughout the project. Bear in mind that the fundamental rule of all business metrics—measure the thing that you actually want to control—applies equally to development project metrics. For example, if you want to improve code quality, don't measure the number of bugs fixed. Measure the number of bugs remaining. When the true goal is hard to measure, it is often tempting to substitute a more easily quantified proxy that you believe tracks the true goal. For example, your true goal might be to improve overall ROI, but you choose to measure costs instead. Hard experience has shown that inevitably people will focus on the thing that is actually measured. Over time the proxy will become increasingly decoupled from and thus unreliable as a measure of the true goal. Worse yet, the greater the rewards or punishments associated with the proxy measure, the faster this divergence will happen. It is better to roughly measure the true goal than to precisely measure a proxy.

Continuous project monitoring

You can regularly monitor the health of the IT projects within your organization through automated metrics collection, project milestone reviews, postmortem reviews and even word of mouth. Continuous project monitoring, combined with iterative development, enables fact based governance by providing up-to-date and accurate metrics based on delivered code versus assessments of specifications. This monitoring enables earlier problem detection, allowing you to take corrective actions sooner and enabling effective governance when you monitor the right metrics. IBM suggests that you begin by automatically capturing and displaying your metrics via dashboard software. At http://jazz.net you can see examples of live project dashboards for the various Jazz product teams, dashboards which display information generated by the development tools being used by the project teams themselves. Furthermore, it's possible to roll up project-focused metrics to the portfolio level, and drill down into the details as needed [28]. But, don't try to manage by the numbers alone. When a project seems to be deviating from its expected course, you should talk with the project team to determine what is actually happening and whether team members need help from you.

Practices for roles and responsibilities

The two lean practices in the roles and responsibilities governance category focus on enabling development by making it clear who is responsible, who has authority and to whom these people are accountable.[19]

Promote self-organizing teams

The first value of agile software development is to foster individuals and interactions over processes and tools.[20] When it comes to lean development governance, the focus should be on enabling IT professionals to build high-quality working software and promoting effective collaboration among team members, instead of trying to control or directly manage them.

A self-organizing team has the authority to allocate the work that it will perform within the scope of the governance structure in which it operates. The team assumes the responsibility of doing the work the way it chooses. Team members select their own activities, everyone commits to the work and the team coordinates regularly. This is a participatory approach to decision making in which everyone has the opportunity to provide input and listen to the decision-making process.

Self-organization works well when combined with iterative development because the iterative approach allows the team and its stakeholders to agree on what should be delivered in the next iteration. At the end of the iteration, the team and all stakeholders assess what was done, and corrective actions are taken. Although the team takes collective responsibility, the ultimate responsibility and associated decision rights lie with the manager. Iterations provide a control mechanism to keep self-organized teams from going off in undesirable direc¬tions. Benefits from self-organization include:

- Higher motivation that can lead to increased productivity
- Decisions being made at the right place within the organizational structure because teams are empowered with both the responsibility and the authority to get the job done
- Fewer opportunities for errors because of better communication and fewer transitions of work products between individuals.
- Greater opportunities for people to increase their skills.

In short, project teams must have the authority to organize themselves, their work environment and their overall approach as they see fit to effectively achieve project results. Any constraints placed on the team, such as organizational guidance (eg. coding standards, data guidelines, security guidelines, and usability guidelines), must be described to and negotiated with the team throughout the project. Our experience is that effective agile teams balance self-organization with appropriate governance, something which is built into IBM's Disciplined Agile Delivery (DAD) process framework.

Align team structure with context

It is important to align team structure with the context of the situation in which the team finds itelf. Conway's law, defined by Melvin Conway in the late 1960s,[21] tells us that any piece of software reflects the organizational structure of the group that produced it (or succinctly, "You ship your organization"). People will work in a manner that reflects the way they are organized. In other words, a decentralized group is likely to produce a system with a decentralized architecture. Any strengths or weaknesses in a project team's organizational structure will inevitably be reflected in the resulting solution that it produces.

For large delivery teams there are four fundamental strategies, which can and often are combined, for organizing their structure:

1. Component teams - With this strategy a subteam is responsible for one or more components of the technical architecture. A component may be a large scale domain component, such as payment processing within a manufacturing company, a collection of web services, a technical framework (i.e. a security framework), or a subsystem. This strategy works well when the architecture of the overall system is highly cohesive and loosely coupled and when the requirements are easily assigned to a single component (large requirements which span components would need to be divided into small requirements which do not).

2. Feature teams - A feature team is responsible for fully implementing a given requirement, the feature, regardless of the number of components the change affects. This strategy works well in environments with developers with a comprehensive knowledge about the system the are working on, a sophisticated change and control process, and a continuous integration approach.'

3. Internal open source - The basic idea is that the artifacts for a component are made available internally within your organization in an open source manner and people from various development teams will evolve the artifacts appropriately. Within IBM Software Group we've applied this strategy for several common components which needed to evolve quickly to support the needs of several disparate teams who wanted to reuse the components. We found that in such situations a component team became overwhelmed with change requests and couldn't easily support the required rate of change. This strategy work well in organizations with an existing and strong open source culture.

4. Specialist teams - A specialist team focuses on the work required for a single development discipline, such as programming, testing, or architecture. Each subteam does their portion of the overall work to implement the required functionality and then hands off to the next subteam(s) down the line. This strategy works well for organizations which still have IT departments comprised mostly of specialists. The authors do not recommend this strategy for anything but a last resort due to the significant bureaucratic burden of organizing such teams, of validating the work when it is handed-off between teams, the increase traceability burden, and other similar problems.

Practices for policies and standards

The three lean practices underlying governance policies and standards describe specific guidelines to support consistent operation among the various parties involved with development.[22]

The first value of agile software development is to prefer individuals and interactions over processes and tools. This doesn't imply that you won't have any processes and tools at all, it simply states that they should be secondary to people, and the tools and processes should reflect the way that people work, not the other way around. With a lean approach to governance your organization will adopt policies and standards that enable developers to collaborate effectively, that motivate them to develop and reuse existing infrastructure and assets, and that enable them to do high-quality work. If there is a conflict between effective individual behaviors and tool behaviors, the tool should concede.

Integrated lifecycle environment

Software development relies on intense collaboration among team members at separate sites who frequently work different hours or in different time zones. The sheer size of many development efforts also makes collaboration difficult because you need to support large-scale interactions. Added to the challenge is the need for management oversight and the necessary bookkeeping to ensure regulatory compliance. Integrated lifecycle environments are built to enable collaboration at the lowest possible cost enabling infrastructure for most if not all the other governance practices.

Key components of an integrated lifecycle environment include tools for software configuration management, analysis, design, construction, test, quality assurance, process management and project management. Lean integrated lifecycle environments such as the IBM Rational Application Lifecycle Management (ALM) solution, facilitate implementation of all or almost all of the identified governance practices. They help you lower your total cost of ownership (TCO) for tool environments, and enable collaboration. And, the best tool environments will automatically capture as much audit and metric data as possible with as little drag on developers' work as possible.

Valued corporate assets

An IT asset is any software component, service, template, enterprise model, reference architecture, example, pattern, guideline or standard that the organization expects IT professionals to apply in their everyday work. A valued IT asset is one that IT professionals actually want to apply, typically because they are viewed as both relevant to the task at hand and of sufficiently high quality. In other words, IT professionals take advantage of the IT assets available to them because the assets actually enable them to increase their productivity.

Developers follow the enterprise architecture and appropriate reference architectures because those assets save them time. Their systems invoke existing services, so they can take advantage of existing technical infrastructure. The key is to make it easier to reuse assets than to build them again.

There are significant organizational benefits to supporting valued IT assets, including increased consistency and efficiency, improved time to market, improved communication and decreased governance costs. IT departments should maintain coding standards for their primary development languages, a library of common patterns that they wish to promote within their systems, reference architectures for critical aspects of their technical infrastructure, and a description of the vision for their enterprise architecture. There should also be people in place to support, evolve, and purchase or harvest these valued IT assets. An asset repository, such as IBM Rational Asset Manager (RAM), is critical to enable IT professionals to find and manage the reusable assets available to them, and to capture and enforce policies related to asset creation and reuse. And, it's important to have metrics in place that can identify and reduce barriers to reuse, as well as measure the value of different elements of the reuse effort.

Wherever possible you should buy rather than build an IT asset, particularly if you can easily acquire something that meets your needs, and thereby put the maintenance burden on someone else.

Flexible architectures

One of the aims of lean software development governance is to enable flexibility and responsiveness in your business, which in turn requires flexibility and responsiveness in how you organize and evolve your IT systems. Lean governance is enabled by flexible architectures because they enable development teams to react effectively to the changing needs of the business. Currently the most flexible approaches to architecture are through open computing and service orientation and through the adoption of lean development techniques such as Agile Model Driven Development,[23] Test Driven Development (TDD),[24] and continuous integration.[25] Flexible architectures can support evolving business needs, reduce time to market, increase ROI, lower TCO and reduce technical risk. By default, IBM recommends service-oriented architectures (SOAs), open source technologies such as Eclipse, and standards such as Java™ Platform, Enterprise Edition (Java EE) and Unified Modeling Language (UML)..

Conclusion

Every IT organization has a development governance program in place, but this program may not be explicit and it may not be effective. Traditional development governance frameworks have had limited success because developers are intellectual workers who don't work well under command-and-control approaches. To be effective, a governance program must reflect the actual environment in which it is being applied, and take into consideration the people involved.

Successful development governance focuses on enabling the right behaviors and the best practices through collaborative and supportive techniques. It proves to be far more effective to motivate people to do the right thing instead of forcing them to do so. The 18 practices IBM has identified for lean development governance reflect the realities of software development today. These practices can enable you to embed governance in your tools, processes and development guidance. They support such modern approaches to development as Lean Software Development, Scrum, XP, Rational Unified Process (RUP), Disciplined Agile Delivery (DAD), and Eclipse Way.

Development governance is only part of the overall IT and systems engineering governance picture, but it's an important one. By streamlining the flow of work performed by the team, a lean approach to governance can help you improve the value provided by development projects.

For more information

To learn more about the Rational tools that can help you implement lean development governance practices, contact your IBM representative or visit:
ibm.com/rational/agile

For an all-in-one lean development solution, learn more about IBM Rational Team Concert at:
ibm.com/rational/rtc

Authors

Scott W. Ambler, Chief Methodologist for Agile and Lean, Rational Software, IBM Software Group

Per Kroll, Chief Solution Architect, Measured Improvements, Rational Software, IBM Software Group

Endnotes

We'd like to thank Cheri Bergeron, Lynn Brennan, Murray Cantor, Bernie Coyne, Clay M. Nelson, Bob Noble, Ted Rivera and Carl Zetie for the feedback and insights they provided for this paper.

[1] Kroll, Per; "Making Agile Mainstream: Crossing the Chasm"; *IBM Rational Edge*; http://www.ibm.com/developerworks/rational/library/mar07/kroll; 2007

[2] Cantor, Murray and Sanders, John D.; "Operational IT Governance"; *IBM Rational Edge*; http://www-128.ibm.com/developerworks/rational/library/may07/cantor_sanders/index.html; May 2007

[3] Poppendieck, Mary and Poppendieck, Tom; *Implementing Lean Software Development: From Concept to Cash;* Addison-Wesley Professional; Boston; 2006

[4] Ambler, S. W. and Kroll, Per; "Best practices for lean development governance—Part 1: Principles and Organizations"; *IBM Rational Edge*; http://www.ibm.com/developerworks/rational/library/jun07/kroll; 2007

[5] Kroll, P. and MacIsaac, B.; *Agility and Discipline Made Easy—Practices from OpenUP and RUP*; Addison-Wesley Professional; Pearson Education, Inc.; Boston; 2006

[6] Bittner, K. and Spence, I.; *Managing Iterative Software Development Projects;* Addison-Wesley Professional; Boston; 2006

[7] Cantor, M.; "Requirements Analysis and Design Part 3"; *IBM Rational Edge*; http://www.ibm.com/developerworks/rational/library/content/RationalEdge/oct03/m_rupse_mc.pdf; 2003

[8] Nightingale, J. ; "Green Threads: Improving Cross-Product Integration"; *Dr. Dobb's Journal*; http://www.ddj.com/architect/196603524; 2006

[9] Ambler, S. W. and Kroll, P.; "Best practices for lean development governance—Part 1: Principles and Organizations"; *IBM Rational Edge;* http://www.ibm.com/developerworks/rational/library/jun07/kroll/; 2007

[10] Kroll, P. and Royce, W.; "Key Principles for Business-Driven Development"; *IBM Rational Edge*; http://www.ibm.com/developerworks/rational/library/oct05/kroll; 2005

[11] Ambler, Scott W.; "Examining the "Big Requirements Up Front (BRUF)" Approach"; http://www.agilemodeling.com/essays/examiningBRUF.htm; 2005

[12] Ambler, Scott W.; "Agile on a Fixed Budget"; *Dr. Dobb's Journal*; http://www.ddj.com/architect/196603524; 2006

[13] Ambler, S. W. and Kroll, Per; "Best practices for lean development governance—Part 2: Processes and Measures"; *IBM Rational Edge*; http://www.ibm.com/developerworks/rational/library/jul07/kroll_ambler; 2007

[14] Agile Alliance; *The Agile Manifesto*; http://www.agilemanifesto.org; 2001

[15] Cantor, Murray and Kruchten. P.; *Rational Unified Process Made Easy—A Practitioner's Guide to the RUP*; Addison-Wesley Professional; Pearson Education, Inc.; Boston; 2003

[16] Kroll, P. and MacIsaac, B; *Agility and Discipline Made Easy—Practices from OpenUP and RUP*; Addison-Wesley Professional; Pearson Education, Inc.; Boston; 2006

[17] Kerth, Norman L; *Project Retrospectives: A Handbook for Team Reviews*; Dorset House; New York; 2001

[18] Ambler, S. W. and Kroll, Per; "Best practices for lean development governance—Part 2: Processes and Measures"; *IBM Rational Edge*; http://www.ibm.com/developerworks/rational/library/jul07/kroll_ambler; 2007

[19] Ambler, S. W. and Kroll, Per; "Best practices for lean development governance—Part 3: Roles and Policies"; *IBM Rational Edge*; http://www.ibm.com/developerworks/rational/library/aug07/ambler_kroll/; 2007

[20] Agile Alliance; *The Agile Manifesto*; http://www.agilemanifesto.org; 2001

[21] Conway, M. E.; "How Do Committees Invent?"; *Datamation Magazine*; http://www.melconway.com/research/committees.html; 1968

[22] Ambler, S. W. and Kroll, Per; "Best practices for lean development governance—Part 3: Roles and Policies"; *IBM Rational Edge*; http://www.ibm.com/developerworks/rational/library/aug07/ambler_kroll/; 2007

[23] Ambler, S. W.; *The Object Primer 3rd Edition: Agile Model Driven Development with UML 2*; Cambridge University Press; New York; 2005

[24] Astels, D.; *Test-Driven Development: A Practical Guide*; Addison Wesley; Upper Saddle River, New Jersey; 2003

[25] IBM; Rational Build Forge; http://www-306.ibm.com/software/awdtools/buildforge; 2007

[26] Ambler, S.W.; *"The Agile Scaling Model (ASM): Adapting Agile Methods for Complex Environments"*; ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/raw14204usen/RAW14204USEN.PDF; 2009

[27] IBM; IBM Practices Plugins for Rational Method Composer; http://www.ibm.com/support/docview.wss?rs=2360&uid=swg24024567; 2010

[28] IBM; Rational Executive Dashboard; http://jazz.net/projects/rational-insight/exec-dashboard/; 2010

[29] IBM; IBM Rational Asset Manager; http://www.ibm.com/developerworks/rational/products/ram/; 2010

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
January 2011

RAW14000-USEN-01