Management Brief

Myths of Methodology

Dispelling some commonly held beliefs about the software development process and IT innovation.

January 2005

Author: Dan Drislane

frontier-strategies.com
info@frontier-strategies.com

This paper originally published in a slightly different form at Enterprise Agility, Inc. (August 2002). enterprise-agility.com

Frontier Strategies, Inc.
1106 West Park Street, Suite 444, Livingston, Montana 59047-2955 USA
Phone: (+01) 248-207-9020 Fax: (+01) 406-794-0506
E-mail: info@frontier-strategies.com
frontier-strategies.com

Introduction

I have wanted to write a paper on methodology for years. Since 1982, when I graduated from college, I have been inundated by the M-word. While working at General Electric, there were a half-dozen methodologies for all manner of engineering and manufacturing, though to be fair, no software development methodologies were in use at my facility at the time. During my seven years at Digital Equipment Corporation, I used at one time or another: three software development methodologies; a Yourdan-based analysis methodology; a business process engineering methodology called TOP; a couple of software integration platforms that just screamed for a methodology (for which I didn't have to wait long); and a host of vendor- or client-mandated standards, including IDEF (the U.S. Air Force), General Motors' and EDS' methods. In my three years at Sun, though I wasn't directly involved in software development during this time, I got a liberal dose of the emerging methodologies of the time, including Ernst and Young's Fusion, Andersen Consulting's Method/1, Ford's proprietary life-cycle methodology, and the revised and enhanced IDEF specs (including how defense contractors were "elaborating" these). These efforts hatched years before object-oriented initiatives—including today's quite popular Rational Unified Process—gained momentum and the Software Engineering Institute at Carnegie Mellon weighed in on best practices and methodology—or what I euphemistically call the *Big M*.

So it was this collective alphabet soup of how to do something in an "organized manner and still be confused" that inspired me to think about methodology from the practitioner perspective. Though I was certainly convinced over the years that using a methodology to design and produce something—a turbine, a car, a software application—was a good idea, I was continually disappointed that they were so hard to understand and implement successfully. During those days with DEC and Sun, with the exception of the well-meaning corporate evangelists that would drop in to give us the religion, my methodology exposure, so keenly observed in colleagues and clients, was more punctuated with dread, denial and cynicism than gun-ho passion. Complaints about methodology were rampant: *"It's too much documentation." "Who has time to read the manuals?" "By the time we do it right, out customer will be out of business."* Why could such a great idea go bad so often?

This paper is about why software development methodologies don't work and why they sometimes do, including a few ideas about how to turn around your own methodology efforts. Often, methodologies don't work because managers and IT professionals hold onto beliefs, practices and organizational paradigms that no longer fit the modern software development organization and the processes that help support it. I call these miscues the *myths of methodology*.

Methodologies don't work because managers and IT professionals hold onto beliefs, practices and organizational paradigms that no longer fit the modern software development organization and the processes that help support it.

I make no claims that I am a methodology expert or a software development guru. Many of my observations are simply from twenty years of personal experience. Computer scientists or genuine methodologists might decide my findings are over-arching. In defense, my only claim is that I am a software professional trying to practice my craft, and hence, a simple robust software development methodology that will make my job easier and my clients more successful is the meal ticket I want. So if I haven't convinced you to drag-and-drop this paper into the recycle bin, then perhaps a few of the pragmatic tips that follow may help you successfully implement your own Big M.

Myth 1: Methodology as Best Practice

Here, we have a crisis of terminology. Why should you care whether a software development methodology is a best practice? It's just different terminology, you say? For one, methodology is not a best practice in and of itself. It is the collected sum of best practices that, when carefully orchestrated, create synergistic, tangible value to an organization. Methodology is also a business process that an organization executes. The process is a collection of best practices (call them *methods* or *tasks* if you wish) that are organized to produce value at one or more points along the *process path*, or if you've been reading the latest management texts, the *value chain*. If you still think I am mincing words here, then consider how you might interpret an engineer from Ford Motor Company telling you that *Job 1* is their vehicle development best practice. Such a statement tells you nothing about how Ford builds cars. To learn about vehicle development and assembly, you'd have to peel back the *Job 1* skin and look at

Figure 1: A successful IT methodology is usually a collection of successful best practices that work together to yield value to the organization. The practices listed above are typical candidates organizations may choose, but aren't an exhaustive list.

some of the dozens of work practices—engineering, manufacturing, and assembly *best practices*—that, together, comprise the *Job 1* methodology. Though Job 1 has been the anthem for Ford for a good decade now, it has evolved and changed as new technologies and better tools and methods for designing and building cars have proven themselves. In essence, Ford is constantly reinventing its methodology as work practices are graduated into best practices.

So, why is this distinction important for the IT organization? Because too often IT folks treat methodology as a singular way of doing business when they should be looking at the smaller scale best practices that comprise the larger methodology, as illustrated in Figure 1. The idea here is not unlike a sports franchise buying the talent needed to produce a winning record and get a shot at the playoffs. Great pieces make for an excellent whole. Proven best practices can make for a successful methodology. Instead of adopting a ready-to-use methodology (we will discuss why this is a bad move), IT organizations should focus on developing competency in best practices first. There are sound reasons for doing this.

For one, lifecycle methodologies are fairly involved mechanisms that are difficult to conquer all at once. Developing and mastering a business process, from the user's vision to delivery of an application, can be cumbersome and unwieldy. It is difficult enough having an experienced development team keep their heads on straight during any one "movement" of a project without having to learn the entire "symphony" that is the lifecycle methodology. Tackling smaller best practices—say, *Design Review* and *Configuration Management*—is a better bet.

Second, most software development best practices focus on a type of work for a certain role or two in the team. For example, *Structural Modeling* might be performed by a business or system analyst, while *Roundtrip Engineering* is usually the providence of the programmer-analyst. This natural tendency toward role and responsibility in the software development lifecycle makes learning of role-aligned best practices easier and more contained. Sure, there are best practices that occupy a broader swath of the lifecycle—*Iterative & Incremental Development* and *Architecture Conformance Validation* (from Figure 1) are two obvious examples—but they can be implemented in phases.

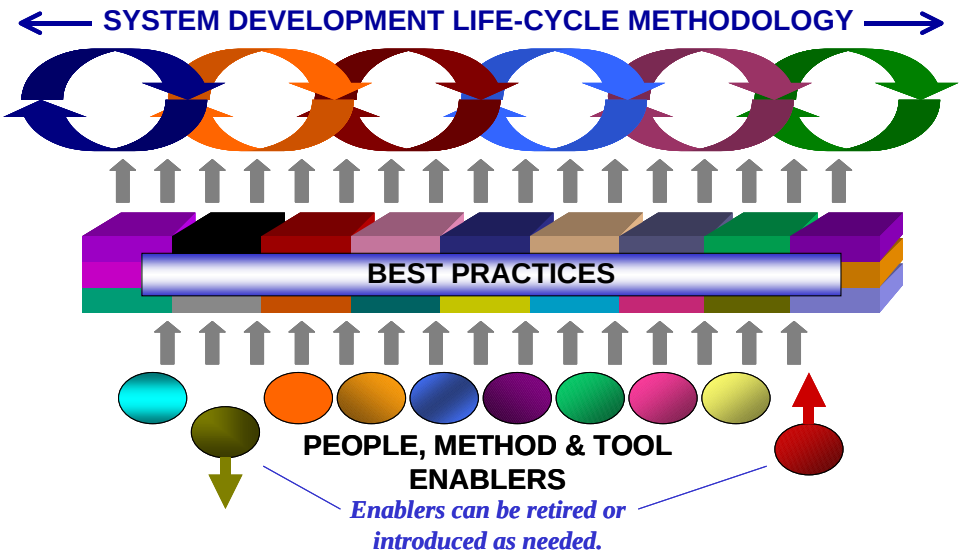

Figure 2: Methodology depends on best practices that are enabled by people, methods and tools.

Third, many of today's software development best practices are more directly coupled to software productivity tools and other advances in technology. A good example of this trend is the popular strategy of implementing an *n-tier architecture*. Separation of concerns and independence of technology, two benefits of n-tier, can be applied to methodology. As a business process, software development methodology can be functionally and organizationally insulated from the tools and techniques that enable the best practices that together work to enable the methodology. Figure 2 illustrates that, as new tools are introduced into the marketplace or developed in-house, they can be deployed as enablers to one or more best practices. A recent example of this is Rational Software Corporation's introduction of XDE[1], which enables programmers to perform roundtrip engineering quickly and easily. A company building their methodology around the Rational Unified Process (RUP) can now evaluate XDE as a likely *Roundtrip Engineering* enabler.

[1] XDE = Extended Development Environment. For more information, go to www.ibm.com/software/rational/.

Figure 3: Building your methodology one best practice at a time. As a best practice proves itself through actual project use, you should implement a business process to "graduate" the practice into your methodology.

However you choose to implement a software development methodology, you'll be far more successful tackling the building blocks—best practices—one at a time. As methods are proven and elevated to best practice status, they can be graduated into the methodology fold. Figure 3 shows how best practices might be graduated into an SDLC methodology. (The role of the methodologist in this scenario is discussed later.)

TIPS

- *Break down your software development methodology into more manageable best practices.*
- *Master best practices first, then graduate and integrate them into the larger methodology.*
- *Evolve your methodology by enhancing existing best practices, introducing new best practices.*
- *Appoint stakeholders that can shepherd and maintain each best practice. Don't overburden one person with more best practices than they can handle; encourage organizational participation and skills development by recruiting staff to adopt a best practice.*

Myth 2: Toe the Line and Conform

If there were a single factor in a methodology failing to gain acceptance in an organization, it is the tendency to resist conforming to a rigid dogmatic set of rules, methods and standards. I can't tell you how many times I have either worked for a company or consulted with clients who have dutifully shown me the corporate IT methodology. There, ensconced on a shelf in all its glory, dressed up in custom printed three-ring binders, would sit the holy grail of the IT organization, a lumbering volume of specifications that documented in incredible detail every facet of specifying, designing, building, testing and maintaining software. And depending on who would show you the goods, this introduction would be accompanied by either an air of smug reverence (the "supporter"), or a smirk and roll of the eyes (the "doubter"). Perhaps you've had a different experience, but in twenty years of working in the IT industry, I have never encountered more supporters than doubters. Still, even today, some large organizations insist on defining an obscenely detailed set of standards to govern the development of software. As most of you can guess, this is a recipe for dismal failure.

Dogmatic methodologies typically fail for the same reasons, regardless of organization:

- **Centralization:** Large IT organizations that service large companies have a tendency to run their business through centralized management and standards. It is not uncommon to have dedicated process, methodology and quality groups. Publishing a corporate IT bible is a sincere attempt to get everyone on the same page. Problems arise because the myriad business groups across the company (often worldwide) have differing work practices, standards and IT skill levels, not to mention cultural values. The team that takes on such an ambitious unification effort cannot possibly consider every cultural and process nuance—the methodology would take four three-inch binders rather than two! The result is that would-be practitioners of the methodology lose faith quickly because the one-size-fits-all approach doesn't speak to them directly.

Would-be practitioners of the methodology lose faith quickly because the one-size-fits-all approach doesn't speak to them directly.

- **Quality Through Quantity:** Many a well-meaning professional thinks that achieving high quality can be had only by documenting *ad nauseum* every possible step and detail of a methodology. It doesn't help matters that influential standards bodies like the International Standards Organization have a legacy of publishing highly detailed treatises; the ISO 9000 quality standards are a good example. The disconnect here is that a software methodology is not a synonym for a detailed specification. To a greater degree, methodology must also embody the business process, the capabilities of the organization (i.e. skill levels and experience) and the business values of the corporation. Any chef who has tried to follow a recipe to the "T" and ends up with a disaster knows perfectly well that cooking is more about skill and knowing your way around the kitchen, which come only from experience, than mindlessly following instructions. Though we all hope that a software development methodology will turn out better applications, publishing a detailed recipe without considering organizational and process factors is not likely to help.

The "Black Car" Problem: Henry Ford's famous quip that customers could have their Model T in any color they wished so long as it was black revealed more than a few things about the idiosyncratic magnate and Ford's company culture. Scratching through the rhetoric, Ford's black car limitation was symptomatic of a highly regimented mass production ethos where even his unskilled workers were considered interchangeable. Producing cars in more than one color would have added complexity, and changes to the production system came from the top, usually Henry himself. Software methodologies

that are centralized and rigid suffer the same "black car" problem. The most effective changes and suggestions for improvements come from the users, those dozens or hundreds of skilled analysts and developers who have to live the methodology every day. Users must be able to influence and be capable of evolving the methodology based on their working experience. A lumbering centralized methodology can't respond to the changing requirements from the field.

TIPS

- *Make it easy for methodology practitioners to provide input; encourage suggestions.*
- *Decentralize the management of the methodology as much as practical.*

Myth 3: One Size Fits All

If you've been part of the IT world for the last decade or more, no doubt you've had some exposure to one or more vendor software development methodologies. For companies like Computer Science Corporation and Andersen Consulting[2] (and before the spin-off craze, its *Big n* competitors), a marquee methodology was a must-have. If you were going after huge development projects, you had to have a methodology to trot into the client's boardroom. Wrapped up in gorgeous die-cut sales brochures and backed by slick slide presentations and high-touch marketing messages, the marquee methodology bolstered credibility and often contributed to justifying large project teams and big budgets. The thinking of course was that following a proven methodology, honed by a world class consulting organization working its magic in the largest Fortune 500 firms, was sound strategy and risk adverse. Though CSC and Andersen have claimed their share of successes using marquee methodologies like Catalyst℠ and Method/1, using such methodologies are a challenge:

- **Huge Body of Knowledge:** Big methodologies are just that—big. Andersen would send their new recruits off to school for six weeks to learn Method/1 and other skills. CSC has a similar program. I worked with Andersen and CSC employees on projects in the 1990s. One problem I noticed was that the sheer immensity of each methodology made it difficult to manage for experienced analysts, let alone novice new-hires. It is challenge enough to keep one's eye on the project while making sure the work and artifacts comply with the methodology. Team members often felt they were really working on two projects: the client initiative, plus the dozens of tasks required to comply with the methodology.
- **Scalability:** The system of integrated checks and balances, those success nuggets so central to a methodology's marketing messages, made it very difficult to downsize Catalyst and Method/1 to smaller projects. Burdened with the overhead of an enterprise methodology, smaller projects create unnecessary complexity in order to comply with the process.

Recently, I participated in a software methodology assessment at a pharmaceutical research center. Though the 110-member IT organization lacked a formal methodology, all parties agreed there was a need for one, but that it must be flexible enough to accommodate two types of applications: (1) enterprise-level business and scientific applications that are high-use/high-visibility; and (2) targeted, small-scale scientific applications that might be thrown away after a few months of use by only a

[2] Andersen Consulting, after divesting completely from Arthur Andersen Worldwide, is now doing business as Accenture.

handful of researchers. What the client correctly discovered is that methodology has to be flexible enough so that its benefits and bother are commensurate with the scope of the business problem for which it is being deployed. They recognized that a *lite* methodology was more suitable for the researcher's needs whereas a more robust process was needed for the enterprise-level projects. This makes good project management and business sense. One user we interviewed during the assessment put it this way: "Why use a sledgehammer to pound a finishing nail?" It is encouraging to see the terms *heavyweight* and *lightweight* methodology surface in web discussions and trade journals lately, a good sign that companies are recognizing the need for an approach based on *right-sizing*.

The *Rational Unified Process (RUP)* from Rational Software Corporation is a good example of a right-sizable methodology and what is gelling as a de facto standard. RUP is more a *meta*-SDLC methodology that can be customized according to the project and team requirements without losing the core benefits of having a sound methodology. In fact, recognizing that the full Monty version of RUP isn't for all organizations, Rational has developed and is marketing an Extreme Programming[3] (XP) strategy for RUP and its modeling tools, such as Rose and XDE, even going as far as publishing a RUP-XP plug-in.

The Development Case: Tweaking RUP

The Rational Unified Process (RUP) was designed as a meta-methodology, one that companies can customize into a SDLC process that fits the specific needs of their business climate and IT culture. Rational calls such customization the *development case*, and provides a web-based front-end to document the standard unabridged RUP, as well as a means to publish the development case.

The development case can use what Rational recommends along the RUP lifecycle, but can also include extensions to RUP as well as customized tools and other enablers that companies wish to incorporate. In doing so, Rational is one of the few commercial software vendors that have recognized that no one methodology will satisfy all user needs.

Though the Rational Unified Process web front-end must be purchased, the RUP itself is well documented by Rational (www.ibm.com/software/rational/) and in the publishing industry (Addison-Wesley has the largest collection of RUP-related titles), and can be implemented to a great degree without purchasing Rational's products. Too, RUP is based on the Unified Modeling Language (UML), a public standard promoted by the Object Management Group (www.omg.org).

TIPS

- *If you're purchasing or adopting a formal methodology, make sure you can strip out components and artifacts that may be overkill for medium to small projects.*
- *Make an earnest attempt to classify your projects by size (small, medium and large) and complexity (easy, moderate, difficult) so you can map the best practices and components of your methodology. One good estimating yardstick for characterizing both project size and complexity is the use-case.*
- *When developing small applications that have limited scope and lifetime, don't sweat the details and overhead of a heavyweight methodology. XP may be a suitable alternative. Judgment and experience are the keys.*

[3] For more information on XP, visit www.extremeprogramming.org.

Myth 4: The Methodologist as MVP

Many companies make the mistake of placing too much importance or responsibility on the methodologist position. Not that I am implying that methodologists play a minor role in organizations implementing a methodology; it's just that, unlike the technical architect, who has to play chief of protocol and enforce technical standards, the methodologist's role is more subtle. There are two basic definitions of methodologist: (1) promoter and keeper of the methodology an organization uses and (2) a person who conceives, designs and implements a methodology. As suggested in Myth 1, a successful IT methodology requires mastery at the best practice level first. The methodology takes shape by incorporating and integrating one or more best practices into a framework that can be utilized and repeated by the organization to accomplish specific goals and produce tangible deliverables. Toward this end, the methodologist is responsible for organizing the best practices, coordinating with working groups, and analyzing the methods for conflicts, glitches and whatever might impede progress. But most important, the methodologist must be a restrained evangelist. A story might help explain the importance of restraint.

Figure 4: Crowning your methodologist Most Valuable Player is usually bad business. All methodologists can only succeed through others in the organization.

While working for Digital Equipment Corporation in the mid-1980s, I was assigned to work with standards efforts surrounding the Manufacturing Automation Protocol (MAP), an ambitious industry-wide effort to standardize communications among computers and factory floor devices, such as robots, assembly machines and programmable logic controllers. The appointed evangelist was the affable Chuck Gardner from Eastman Kodak in Rochester. As the MAP point man, Chuck had his hands full. In addition to promoting and managing Kodak's internal MAP efforts, he chaired a large multi-company working committee, oversaw numerous ad hoc groups, and was liaison to the International Standards Organization as well as the European MAP organization. As such, Chuck was highly visible in the MAP movement and was frequently called to speak at conferences and in boardrooms.

You'd think such a position would have gone to Chuck's head. Much to the contrary, he was one of the most approachable, easy going and accommodating people I've known. Though he had an encyclopedic mind about data communications, knew the workings of a factory inside-out, and had the ears of many executives, he was most comfortable chit-chatting with engineers and technicians on the shop floor, MAP's real proving ground. Chuck advanced his agenda by listening intently, probing for answers and suggestions, and building consensus among myriad organizations, some who directly competed against each other. He knew that only by appealing to real-world constituents—factory employees who would actually use the stuff—would the MAP standard have any chance of succeeding. Chuck was instrumental in championing an extremely detailed set of technologies that were adopted worldwide.

IT methodologists would do well to emulate Chuck Gardner. Less guru and more bandleader, the methodologist should be measured more by how the methodology is being engaged by the organization rather than by how much air play management is giving it. Though there are obvious reasons why you should stabilize a methodology and controls its evolution—a dozen flavors of an SDLC methodology is not a good thing—the goods need to be evaluated, road tested, challenged and improved by the staff members and IT project teams that do the actual work. Like Chuck Gardner, the IT methodologist has to

be a restrained evangelist: part diplomat, part methodology expert, part referee and part management liaison.

TIPS

- *Methodologists don't succeed by pushing down policy and standards, but by developing ideas from the IT organization.*
- *Methodologists are more than process scientists; they are cheerleaders, diplomats and great listeners.*
- *Make sure your methodologist knows he or she is a benevolent dictator working in a healthy democracy. Policy decisions should be thought through carefully.*

Myth 5: Methodology and Architecture As Fiefdoms

Many IT managers seem to lump their development methodology and architectural concerns together under the same roof. Still others tend to do quite the opposite and direct different people or teams to tackle each domain separately. Neither is a wise choice. Methodology and architecture are unique disciplines that must integrate into each other's frameworks. There is good reason, however, to appoint both roles to one person, provided the candidate has a strong architectural background. Let's see why.

Architects have the challenging task of defining, implementing, policing and maintaining an IT application and execution framework. Typically (for there are as many variants of architects as there are companies that employ them), every software and hardware asset being used in development, maintenance and production—from development environments, to mail and web servers, to the accounts receivable tax tables—is under the tutelage of the architect (or architecture group). Architects concern themselves with myriad topics, such as execution and communication frameworks; languages;

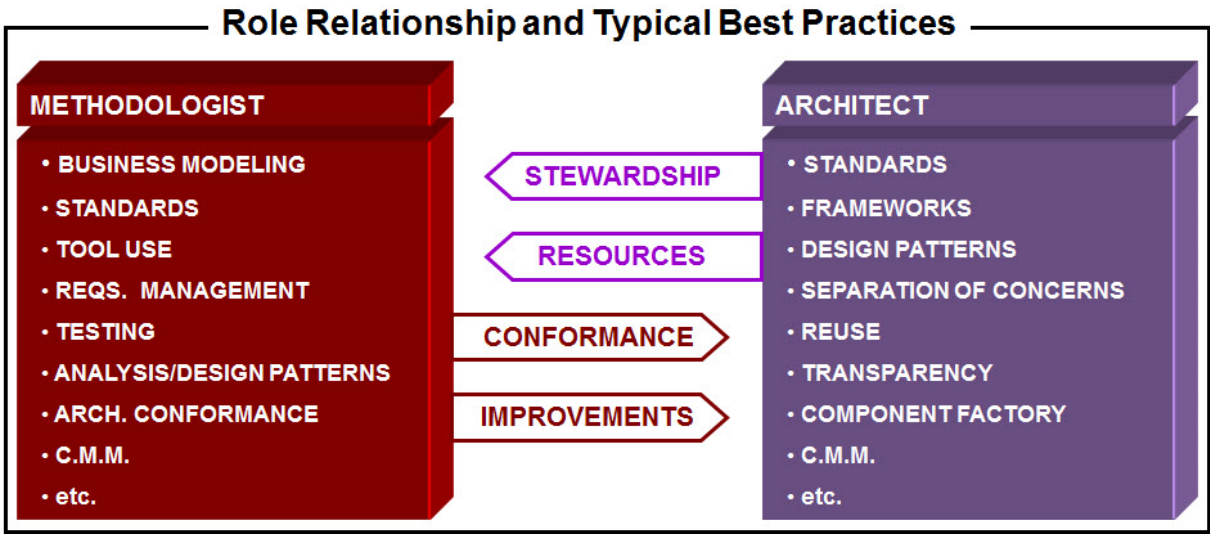

Figure 5: If not the same person, which is desirable, the methodologist and architect should have co-dependent roles, with each helping to develop and refine the other's domain.

naming standards; middle- and groupware; implementation strategy; plus more esoteric topics like separation of concerns; transparency; consistency; viewpoints; reuse and patterns; component factories; and the Software Capability Maturity Model[4].

IT methodologists are no less busy though it could be argued that their domain is relatively smaller since it is usually software development, testing and change management that is the focus. Of typical concern to the methodologist are: business modeling; naming, modeling and code standards; role definitions (who does what in a typical project); tool use; requirements elicitation and management; use case design and standards; teaming and integration; testing methods and regimes; documentation standards; definition of project artifacts[5]; peer reviews (design and code); and configuration management. More esoteric pursuits include: abstraction, refactoring (what and when); structural and behavioral modeling; scenario analysis; analysis and design patterns; facades; subsystems; delegation; evaluation and access to reusable objects and components; architectural conformance; and the Software Capability Maturity Model, among many others.

The fact that there are several interests that architects and methodologists share may tip you off where some of the integration points are and why both disciplines can't be considered strict fiefdoms. In fact, a company's architectural frameworks and the methodologies that help build those frameworks are co-dependent upon one another. The methodologist must develop, structure and integrate the best practices that make up the methodology so that it supports the frameworks established by the architect. On the other hand, the architect must realize that a chief delivery mechanism for correction and improvements to the architecture is the methodology (i.e. the sum of best practices) being deployed by the development, testing and support teams. For example, *n-Tier Design & Implementation* (Figure 1) has direct impact on several architectural considerations: separation of concerns, transparency (both mentioned above), as well as partitioning and independence of technology. Another obvious example (also from Figure 1) is *Architectural Conformance Validation*, a best practice that checks that the artifacts produced by, say, the analysts and developers, comply with the various architectural frameworks. Sure, architecture can stand alone without a formal development or testing methodology, but it will be poorer for it since, aside from the technology that helps define an architectural framework, it is methodology that drives innovation and quality in the overall architecture.

TIPS

- *Except in the largest of organizations, assign the methodologist and architecture roles to the same person, provided that person is a strong architect.*
- *Formalize the dependencies of methodology and architecture and check regularly that these dependencies are valid.*
- *If the roles are separate, involve your architect in methodology development and also projects. Ask to have the methodologist participate in architecture development initiatives.*

[4] Discussion of the Software Capability Maturity Model (SW-CMM) is out of scope for this paper. SW-CMM is developed and maintained by the Software Engineering Institute (SEI), a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University. For more information, visit: www.sei.cmu.edu/cmm/cmms/cmms.html.

[5] Project artifacts, those tangibles that are produced in the course of a development or maintenance effort, are also the project manager's concern.

Myth 6: Attending Methodology University

I am dubious about organized training on methodology because it is too tempting for a company to build a heavyweight program that is overly formal and inflexible. For companies like Andersen, CSC, Ernst & Young and Digital, a trip to the training center was the proven way to indoctrinate employees in the corporate methodology. True, these firms also relied on on-the-job training (OJT) via client projects, but *Methodology U.* was the first stop for new employees, especially entry-level hires. Rather, methodology is best learned through three approaches: certainly OJT training, but also specialization and mentoring.

OJT speaks for itself since most of you have likely learned many software tools you have used in the past and present by clicking away and referencing help files when needed. As discussed in Myth 1, methodology is really an organized collection of best practices. OJT learning of best practices is useful because learners aren't inundated with the entire methodology all at once. They can focus on, say, *Behavioral Modeling* before they tackle *Roundtrip Engineering*. Mastery of the best practices allows the learner to internalize the value of each practice in the context of a real project without being overwhelmed by the larger methodology.

Specialization is merely the recognition that, in a SDLC methodology, not all best practices will need to be mastered across the IT organization. Even with non-waterfall, iterative methodologies, such as the Rational Unified Process, up to 33 unique roles could be staffed for a development project, though a more practical number would be a third of that. It is not necessary for a business analyst to master the best practice *Pair-Programming* just as it wouldn't be critical for developers to be requirements experts. While knowledge of the whole process of producing software applications is desirable, you should expect that people will be inclined to specialize in one or more best practices.

Mentoring is an invaluable tool that too many companies trivialize or ignore altogether. Working hand-in-hand with a person who has mastered a best practice is a valuable experience because it promotes knowledge transfer far more effectively than in the training room. The problem isn't just training. In order to learn a new way of working individuals require an approach where they have regular direct access to a resource to help them through the snafus. The mentor also serves as change agent since it is he or she who will be introducing best practices into the project environment.

A final thought on the value of learning methodology through OJT, specialization and mentoring. Because best practices and their enablers evolve, and consequently methodology does as well, it is quite difficult to maintain an organized training program. Training development is expensive, time-consuming and usually lags behind the introduction of the topical matter (whether it's a new software application or a set of methods). New practices and improved methods can be announced and put into play quickly using OJT and appropriate mentoring from the team that introduced the enhancements.

TIPS

- *Avoid developing and delivering formal classroom training for your methodology.*
- *Employ on-the-job training and mentoring to learn and master the best practices that comprise your methodology.*
- *Recognize that specialization is a natural learning paradigm and that few, if any, IT staffers will master the entire methodology. Encourage mastery of the best practices that fit the role each person is playing in a project.*

Myth 7: Stifling Creativity

While conducting a methodology and best practices review for a client last year, one common refrain we heard in our interviews was that a formal methodology would strangle the developer's creativity in designing, coding and implementing applications. Right or wrong, many felt that the "rules of conduct," as one interviewee jokingly referred to methodology, would hinder innovation and cause undo complexity in building systems. While heavyweight methodologies that are cumbersome to use might be guilty of stifling creativity, a right-sized, flexible methodology based on best practices will not only not repress ingenuity, it is likely to improve it.

The opinions formed by our client weren't unusual or novel. Many developers want a free hand in their work and most associate design, coding and architectural standards as unnecessary roadblocks. Unfortunately, some assume that formal naming standards, the ordering of tasks and modeling techniques, to name three perceived components of methodology, means that their days as crafters of a great design or code are over. Actually, it couldn't be further from the truth. There are two reasons for this:

- **Reason 1:** Methodology helps focus creativity on high business value.

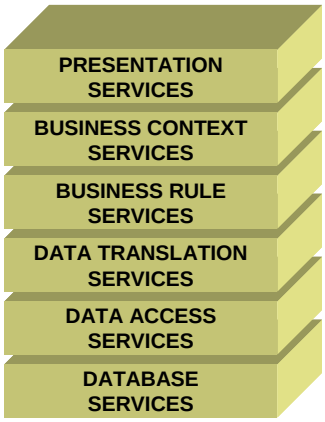

Figure 6: N-tier architectural pattern (in this case, 6-tier) promotes separation of concerns and helps channel creativity into execution layers that map more readily with individual skill sets.

Methodology exists to make software development a repeatable process. Repeatability has some obvious benefits: reduced time to delivery; lower maintenance costs; faster adoption of new best practice; and faster training ramp-up of new employees. However, one additional benefit is that it also aligns the IT organization with the business of the company.[6] This is important because it helps stem IT innovation that adds only marginal value, what I term *commodity creativity,* and instead focuses IT professionals on what will truly benefit the core business of the company. Project team members should no longer be flustered that best practices such as *Architecture Conformance Validation* or *Unified Change Management* check their freedom and imagination. What such best practices do—particularly these two—is remove some of the tedious decisions that managers, designers and developers have to make when they're engaged in a development project. Project contributors are then left to focus more intently on solutions that add more business value and less behind-the-scenes structure.

Still, some of our client interviewees were unmoved: *"Why do we need to conform to standards when our window of opportunity is short and we need to cobble together a quick and dirty application? We need the flexibility to be creative."* Exactly! This is where right-sizing your methodology is important. If you need to develop an application that will be used short-term and has limited exposure, why complicate matters with a heavyweight methodology or even any methodology? If it's throwaway code, who cares? Allow your application developers complete freedom to create what they need for the

[6] Only a small percentage of North American businesses actually produce software as their primary product; the majority of companies are goods and service providers and their IT organization (whether internal or contracted) plays a supporting role.

short-term business at hand, which will add value without suffocating creativity. Just be sure that what is being built won't have an adverse impact on other efforts or your enterprise architecture.

- **Reason 2:** Emerging software development paradigms and architectural frameworks provide a foundation for methodology innovation.

One positive development over the last ten years has been the concept of identifying, developing and exploiting patterns in application development. A *pattern* is merely a proven method or construct that solves a specific problem in a project. As such there are analysis patterns, design patterns and architectural patterns that are gaining popularity with IT practitioners who don't wish to recreate the wheel. Methodology patterns also exist. The basic premise of patterns is that when a group of elements is reduced to its most basic and elegant form, which add value to solving a problem, then a recognized pattern emerges. Establishing patterns are a natural defense against project teams spinning their creative wheels on *commodity problems* that have already been solved. Again, this helps focus creative minds on what is higher value-add to the core business.

Similarly, trends in IT architecture help define best practices that channel opportunities for creativity where they will have the most impact to the business and, in some cases, the supporting technical infrastructure. As just discussed, *N-tier Design and Implementation* can be considered both a best practice and an architectural pattern. N-tier allows the architect to establish *creativity zones* along the business layers or technical support layers of the architecture. (It also allows IT management to organize staff along business lines or in technical support roles.) As Figure 6 shows, the business context and business rule service layers are two creativity zones that project analysts and designers can focus on. Instead of debating how to implement transaction management, the team can be focused on, say, finding business rule patterns. N-tier would also focus technical experts in the lower persistence layers (Data Access Services and Database Services) to develop more innovative service capabilities.

Other architecture practices, such as transparency, consistency and reuse, also help pave the way toward creative thinking and innovation.

TIPS

- *Use best practices and your overall methodology to help you channel your staff's creativity away from low value problems and toward high-value or high-risk problems.*
- *Challenge your staff to associate creativity with solving critical business and technical problems, not just elegant code streams or quick-and-dirty fixes.*
- *Recognize that right-sizing your methodology also contributes to innovation and creativity. Don't strangle ingenuity with a heavyweight methodology.*
- *Exploit the paradigm of analysis, design and architectural patterns to help channel creativity toward unsolved problems and new, undiscovered patterns.*

Myth 8: Wasting Away in Methodologyville

Is methodology a waste of time? To introduce this section, an unscientific comparison of people and society is in order. Methodology can be loosely compared to a society's laws and community standards. Laws and community standards help promote societal values, a code of conduct that the society will tolerate, and a sense of organization and security. A side benefit is that organized societies with a system of laws tend to be more productive and even more innovative.

In much the same manner, an IT methodology helps promote the values of the business society—the IT organization and its customers, the business users—and it also provides a framework that helps organize work and produce artifacts that help deliver the applications and systems the users need. Though this may be a leap of faith for the reader, except for the most radical of thinkers, laws and community standards are not wasteful endeavors but instead serve the common good of the society that enacts them. So goes methodology. For societies of IT teams and the users they serve, a methodology that helps steer IT initiatives in the right direction is probably good. While smaller societies may stand a better chance of governing themselves without the constraints of a formal methodology, some system of methods or standards usually gels in even the smallest of teams. This is the primary reason Extreme Programming (XP) works so well with small teams. It's not that the XP methodology is so innovative or unique; it's just that it exploits the inherent productivity gains of small groups of workers (pairs, in the XP case).

Figure 7: Methodologists must recognize that their mission will benefit from the business organization's laws, community standards and teamwork.

At Digital, one simple exercise we used to request of our clients, usually at the onset of a software project, was to have the people we were meeting with draw their business process. Though most people could readily describe how the division or department worked, it was always surprising how much disparity there was between people's views. At one client, despite the fact that two people worked together every day, the quality manager gave a significantly different account of the production process than the line supervisor. There were usually two culprits at work here: (1) internalization of the process (*"How do I interact with the business process?"*) causing disjointed views, and (2) the lack of a clear, concise and published business process. Both symptoms gave our team pause. If our clients didn't have an accurate grasp on their business, their vision for the project might also be skewed. In these cases, we usually took time to gain consensus on what the business processes were before we ventured into gathering business requirements and proceeding with the project.

IT organizations should ascribe the same level of importance when looking at their business processes and, ultimately, their SDLC methodology. It is time well spent to examine the business strategy of the company as a whole and then the IT organization's business strategy so a vision for an SDLC methodology can be formulated. It is time well spent to examine the organization's core competencies and legacy best practices that can be candidates for inclusion into a broader SDLC methodology. In fact,

it may be valuable to take the time to come up with a *methodology development and deployment process*, a sort of methodology to discover and promote your SDLC methodology!

Time invested thinking up front about what methodology and best practices can buy your organization will pay dividends later, usually in the form of better responsiveness to your customers, lower maintenance costs and faster time to market for your services and deliverables. And unless you take the time to analyze and prove why a right-sized methodology is good for business, you'll be missing the opportunity to establish a societal framework that will help promote innovation, teamwork, understanding and performance.

TIPS

- *Take the time to examine your IT organization's business processes to make sure they support the company's goals and strategy. Only then can you begin to examine your core competencies, best practices and a formal SDLC methodology.*
- *Recognize that developing and evolving a methodology takes time and it's everyone's business.*
- *Look at developing a methodology development and deployment process, one that will help you discover and promote your SDLC methodology.*

About the Author

Dan Drislane is the founder of Frontier Strategies, Inc. in Livingston, Montana, an IT consulting firm begun in 1991 in Michigan. He has over 20 years of experience in business analysis, business process analysis and project management. Dan's work with clients focuses on two goals: transforming the IT organization's culture so it will be more agile and accountable to business; and integrating an organization's vision and supporting business processes with its enterprise business and system architecture.