CHAPTER 1

The Agile Data Method

It is possible to take an agile approach to data-oriented development. The first step is to choose to work this way.

Data is clearly an important aspect of software-based systems — something we've all known for decades — and yet many organizations still struggle with their approach to data-oriented issues within their software processes.

The goal of the agile data (AD) method is to define strategies that enable IT professionals to work together effectively on the data aspects of software systems. This isn't to say that AD is a "one size fits all" methodology. Instead, consider AD as a collection of philosophies that will enable software developers within your organization to work together effectively when it comes to the data aspects of software-based systems. Although the focus of this book is proven techniques for agile software development, it's critical to define an underlying methodological foundation.

In this chapter, I help you understand the AD method by exploring the following topics:

- Why working together is currently difficult
- The agile movement
- The philosophies of agile data
- Agile data in a nutshell
- Does agile data address our problems?

Why Working Together Is Currently Hard

In many organizations, the relationship between data professionals and developers is often less than ideal. Yes, there are some organizations where these two communities work together quite well, but there are always tensions — some healthy tension exists between groups, and from these your organization can benefit, but when the tension isn't healthy these differences often lead to conflicts. The challenges that data professionals and developers must overcome can include:

Different visions and priorities. Developers are often focused on the specific needs of a single project and often strive to work as much as possible in isolation from the rest of the organization. Database administrators (DBAs) focus on the database(s) that they are responsible for, often "protecting" the databases by minimizing changes to them. Data administrators and data architects focus on the overall data needs of the enterprise, sometimes to the virtual exclusion of the immediate needs of project teams. Clearly, the scope of each group is different, their priorities are different, and the issues that the groups deal with are different. To make matters worse, your project stakeholders, including direct users all the way up to senior management, have varying priorities and visions as well.

Overspecialization of roles. Specialists have a tendency to become too narrowly focused; they can work so hard to know everything there is to know about a small slice of software development that they can become oblivious of everything else. For example, it's quite common to find senior Java developers that have never heard about data normalization (discussed in Chapter 4), or even understand why you would want to do such a thing, and data architects who can't read a Unified Modeling Language (UML) state chart diagram (discussed in Chapter 2). Because these roles are overly specialized, the people in those roles often have difficulties relating to others. At the other end of the spectrum are generalists who understand the big picture but don't have any concrete skills to offer a development team. We need to find the sweet spot between these two extremes. An underlying philosophy of *Agile Modeling* (Ambler 2002a) is that software developers should have a general understanding of the overall software process *and* have one or more specialties. Because agile modelers are generalists, they understand the broad range of issues pertinent to the "software game" and yet they still have specific, valuable skills to offer to their team.

Process impedance mismatch. One of the few things that processes such as the Unified Process (Kruchten 2000; Ambler 2001b), Extreme Programming (XP) (Beck 2000), Scrum (Beedle and Schwaber 2001), DSDM (Stapleton 1997), Crystal Clear (Cockburn 2001b), feature-driven development (FDD) (Palmer and Felsing 2002), and Agile Modeling (AM) have in common is that they all work in an evolutionary (iterative and incremental) manner. Unfortunately, many within the data community still view software development as a serial or near-serial process. Clearly, there is an impedance mismatch here, indicating that the data community needs to rethink its approach. You will see in Part II of this book that it is possible to take an evolutionary approach to data, a change that will require cultural and organizational adjustments to succeed.

Technology impedance mismatch. Developers work with objects and components, whereas data professionals work with databases and files. Software-engineering principles form the underlying foundational paradigm for objects and components, whereas set theory forms the underlying foundational paradigm for relational databases (by far the most popular database technology). Because the underlying paradigms are different, the technologies don't work together perfectly, and an impedance mismatch exists. This mismatch can be overcome, although doing so requires a significant skillset (this topic is covered in Chapter 7).

Ossified management. The technology and techniques used by software developers change rapidly, a fact that we all know very well. As people progress up the corporate hierarchy, they deal less with technology and more with people issues, the end result being that many managers have lost their technical edge. The implication is that management's previous development experiences, on which they base technical decisions, may no longer be applicable. We experienced this when we moved from procedural to object-oriented technologies — what may have been a good decision on a COBOL project often proves to be the kiss of death to a Java project. We're clearly seeing this problem once again as we move to agile software processes. Management needs to change with the times.

Organizational challenges. Common problems, such as poor communication or politics among individuals and groups, hurt the data aspects of software development just as badly as they hurt other efforts, by preventing everyone from working together effectively.

Poor documentation. Most documentation seems to be at one of the following extremes: little or no documentation or overly complex documentation that nobody reads. Mutually agreed-to development standards and guidelines, legacy system documentation, legacy database documentation, and enterprise models can be valuable resources when written well. Chapter 10 presents agile strategies for writing documentation.

Ineffective architectural efforts. Most organizations face significant challenges when it comes to enterprise architecture, the most common of which being that they don't know where to start. Biased enterprise architectures that overly focus on one view of the enterprise lead to architectures that do not adequately address the real needs of an organization. As the Zachman Framework (ZIFA 2002; Hay 2003) indicates, there are many potential views that you want to consider. These views are data/structure, function/process, network, people, time, and motivation. Ivory tower architectures — those formulated by teams that have removed themselves from the day-to-day realities of project teams — look good on paper but unfortunately fail in practice. Furthermore, developers need to accept that their efforts must reflect and conform to the constraints imposed on them by their organization's environment.

Ineffective development guidelines. Many organizations struggle to come to a collection of development guidelines that all software developers will work to. There are a large number of causes for this, including people not understanding

the need to follow such guidelines, people unwilling to follow someone else's guidelines, overly complex guidelines, overly simplistic guidelines, a "one size fits all" attitude that leads to inappropriate guidelines for a specific platform, and an unwillingness to evolve guidelines over time. When you have an effective collection of guidelines available to you, and (this is key) *everyone understands and applies them appropriately*, you can dramatically improve the productivity of your software development efforts.

Ineffective modeling efforts. This is often the result of several of the previously identified problems. People focused on a specific aspect of development will often produce models that wonderfully reflect the priorities of that narrow view but fail to take into account the realities of other views. An enterprise data model may present an excellent vision of the data required by an organization, but an enterprise model that reflects the data, functional, usage, and technical requirements of an organization is likely to be far more useful. A UML class diagram may reflect the needs of a single project, but if it doesn't reflect the realities of the legacy data sources that it will access then it is of little value in practice. Modelers, and software developers in general, need to work together and look at the full picture to be truly effective.

Detecting That You Have a Problem

It is very easy for organizations to deny that they have a problem. It can be very difficult for senior management to detect problems until it's too late because the bad news that they need to hear is filtered out long before it gets to them. Similarly, it can be difficult for people elsewhere in the organization to detect problems — perhaps everything is going quite well in their opinion — unfortunately the value system that they're using to judge the situation isn't ideal, making them blind to the problems that they are causing.

As a consultant I have the privilege of working in a wide range of organizations, and it seems to me that about one in ten organizations is reasonably successful with its approach to data-oriented activities, about six in ten think they're doing well but really aren't, and the remaining three in ten know that they have a problem but don't know what to do about it. It doesn't have to be this way.

So how do you know you've got a problem? Enterprise data professionals, including both data architects and data administrators, will be frustrated by the fact that project developers on project teams ignore their advice, standards, guidelines, and enterprise models. Worse yet, application developers often don't even know about these people and things in the first place. Developers will be frustrated by what they perceive (often rightfully so) to be the glacial pace of enterprise data professionals to make or authorize seemingly simple changes. DBAs often find themselves stuck in between these two warring factions, trying to get their work done, while struggling to keep the peace. If one or more of these problems is common within your organization you've got a problem.

The following is a list of potential symptoms that may indicate that your organization has one or more challenges that the agile data method may help you address:

- People are significantly frustrated with the efforts, or lack thereof, of one or more groups.
- Software is not being developed, or if it is it is taking far too long or is much too expensive.
- Finger pointing occurs such that you hear things like "the data administrators are holding up progress" or "the developers aren't following corporate guidelines." Worse yet, the finger pointer typically doesn't perceive that he or she is also part of the problem.
- Political issues are given higher priority than working together to develop, maintain, and support software-based systems.
- Ongoing feuds exist between people and groups. Phrases that start with "you always" and "you never" are good indicators of this.
- Well-known problems within your organization are not being addressed. Furthermore, suggestions for improvements appear to be ignored, nothing happens, and no reason for rejection is provided.
- People are working excessively long hours with little or no reward.
- Decisions affecting teams — in particular project teams — are made in an apparently arbitrary and arrogant fashion.

We need to find a way to work together effectively. There are clear differences between the data and development communities as well as between the project and enterprise communities. The fact that we're talking about different communities is also part of the problem, arguably one of the root causes. You have a fundamental decision to make: Should you use these differences as an excuse to exacerbate existing problems within your organization or should you revel in these differences and find a way to take advantage of them? I prefer the latter approach. My experience is that the values and principles of the agile movement form the basis for an effective approach to working together.

The Agile Movement

To address the challenges faced by software developers an initial group of 17 methodologists formed the Agile Software Development Alliance (www.agilealliance.org), often referred to simply as the Agile Alliance, in February 2001. An interesting thing about this group is that the members all came from different backgrounds, and yet they were able to come to an agreement on issues that methodologists typically don't agree upon (Fowler 2001a). This group of people defined a manifesto for encouraging better ways of developing software, and then, based on that manifesto, formulated a collection of principles that defines the criteria for agile software development processes such as AM.

The Manifesto for Agile Software Development

The manifesto (Agile Alliance 2001a) is defined by four simple value statements — the important thing to understand is that while you should value the concepts on the right-hand side, you should value the things on the *left-hand side* even more. A good way to think about the manifesto is that it defines preferences, not alternatives, encouraging a focus on certain areas but not eliminating others. The Agile Alliance values are as follows:

Individuals and interactions **over processes and tools.** Teams of people build software systems, and to do that they need to work together effectively — teams include but are not limited to programmers, testers, project managers, modelers, and customers. Who do you think would develop a better system: five software developers with their own tools working together in a single room or five low-skilled "hamburger flippers" with a well-defined process, the most sophisticated tools available, and the best offices money could buy? If the project were reasonably complex, my money would be on the software developers, wouldn't yours? The point is that the most important factors that you need to consider are the people and how they work together; if you don't get that right the best tools and processes won't be of any use. Tools and processes are important, don't get me wrong, it's just that they're not as important as working together effectively. Remember the old adage, *a fool with a tool is still a fool*. This can be difficult for management to accept because they often want to believe that people and time, or men and months, are interchangeable (Brooks 1995).

Working software **over comprehensive documentation.** When you ask a user whether he or she would want a fifty-page document describing what you intend to build or the actual software itself, what do you think that person will pick? My guess is that 99 times out of 100, the user will choose working software, assuming of course that he or she expects that you can actually deliver. If that is the case, doesn't it make more sense to work so that you produce software quickly and often, giving your users what they prefer? Furthermore, I suspect that users will have a significantly easier time understanding any software that you produce than complex technical diagrams describing its internal workings or describing an abstraction of its usage. Documentation has its place, written properly, it is a valuable guide for people's understanding of how and why a system is built and how to work with the system. However, never forget that the primary goal of software development is to create software, not documents — otherwise, it would be called documentation development wouldn't it?

Customer collaboration **over contract negotiation.** Only your customer can tell you what they want. No, they likely do not have the skills to exactly specify the system. No, they likely won't get it right at first. Yes, they'll likely change their minds. Working together with your customers is hard, but that's the reality of the job. Having a contract with your customers is important, but, while having an understanding of everyone's rights and responsibilities may form the foundation of that contract, a contract isn't a substitute for communication. Successful developers work closely with their customers, they invest the effort to discover what their customers need, and they educate their customers along the way.

Responding to change **over following a plan.** People change their priorities for a variety of reasons. As work progresses on your system, your project stakeholders' understanding of the problem domain and of what you're building changes. The business environment changes. Technology changes over time and *not* always for the better. Change is a reality of software development, a reality that your software process must reflect. There is nothing wrong with having a project plan; in fact, I would be worried about any project that didn't have one, but a project plan must be malleable; that is, there must be room to change it as your situation changes; otherwise, your plan quickly becomes irrelevant.

The interesting thing about these value statements is that almost everyone will instantly agree to them, and yet rarely do people adhere to them in practice. Senior management always claims that its employees are the most important aspect of the organization, and yet often they follow ISO-9000-compliant processes and treat their staff as replaceable assets. Even worse, management often refuses to provide sufficient resources to comply with the processes that they insist project teams follow — the bottom line: management needs to eat its own dog food. Everyone will readily agree that the creation of software is the fundamental goal of software development, yet many people still insist on spending months producing documentation describing what the software is and how it is going to be built instead of simply rolling up their sleeves and building it. You get the idea — people often say one thing and do another. This has to stop now. Agile modelers do what they say and say what they do.

The Principles for Agile Software Development

To help define agile software development, the members of the Agile Alliance refined the philosophies captured in their manifesto into a collection of 12 principles (Agile Alliance 2001b) that methodologies, including agile data (AD), should conform to. These principles are:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter time scale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

- Continuous attention to technical excellence and good design enhances agility.
- Simplicity — the art of maximizing the amount of work not done — is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective and then tunes and adjusts its behavior accordingly.

Stop for a moment and think about these principles. Is this the way that your software projects actually work? Is this the way that you think projects should work? Reread the principles again. Are they radical and impossible goals as some people would claim? Are they meaningless motherhood and apple pie statements? Or are they simply common sense? My belief is that these principles form a foundation of common sense upon which you can base successful software-development efforts, a foundation that can be used to direct the data-oriented efforts of software developers.

The Philosophies of Agile Data

First and foremost, the agile data method subscribes to the values and principles of the Agile Alliance. Although this advice is a very good start, it needs to be extended with philosophies that reflect the realities faced by data professionals. The philosophies of agile data are:

Data. Data is one of several important aspects of software-based systems. Most, if not all, applications are based on moving, utilizing, or otherwise manipulating some kind of data, after all.

Enterprise issues. Development teams must consider and act appropriately regarding enterprise issues. Their applications must fit into the greater scheme of things by conforming to the common enterprise architecture (or at least to the future agreed-upon architecture), by following common development standards, and by reusing existing legacy assets wherever possible.

Enterprise groups. Enterprise groups exist to nurture enterprise assets and to support other groups, such as development teams, within your organization. These enterprise groups should act in an agile manner that reflects the expectations of their customers and the ways in which their customers work.

Every project is unique. Each development project is unique, requiring a flexible approach tailored to its needs. One software process does not fit all, and therefore the relative importance of data varies based on the nature of the problem being addressed.

Teamwork. Software developers must work together effectively, actively striving to overcome the challenges that make it difficult to do so.

Sweet spot. You should actively strive to find the "sweet spot" for any issue, avoiding the black and white extremes to find the gray that works best for your overall situation.

Interestingly, most of these philosophies aren't specific to data; instead, they are applicable to software-development efforts in general. As the first principle implies, you need to look at the overall picture and not just data; therefore, data-specific principles very likely won't serve you very well. Heresy? No. Just common sense.

Agile Data in a Nutshell

The best way to understand the AD method is to explore its four roles — agile DBA, application developer, enterprise administrator, and enterprise architect — and how they interact with each other. The first two roles, the focus of this book, are project-level development roles. The second two roles, which are featured prominently throughout the book due to their importance, are enterprise-level support roles. An agile software developer can take on one or more of these roles, although his or her focus will very likely be on one or both of the project-level roles.

Let's explore each role in greater detail.

Agile DBAs

An *agile DBA* (Schuh 2001) is anyone who is actively involved with the creation and evolution of the data aspects of one or more applications. The responsibilities of this role include, but are not limited to, the responsibilities typically associated with the traditional roles of database programmers, database administrators (DBAs), data testers, data modelers, business analysts, project managers, and deployment engineers. This is the type of role that a DBA within a small organization typically finds himself or herself in: a sort of "data jack of all trades."

The primary customers of agile DBAs are application developers, although enterprise administrators and enterprise architects are very close seconds. When agile DBAs are asked to support the business community, their primary customers will also include direct end users and their managers. This is particularly true when agile DBAs support applications that are data focused, in particular reporting applications.

An agile DBA will work closely with application developers, typically supporting a single larger team or several smaller teams as the case may be. Agile DBAs can often be responsible for several data sources (for example, databases, files, XML structures, and so on) or at least be coresponsible for them. For example, if two development teams access the same database and each of them has its own agile DBA, those two people will then need to work together to evolve that database over time. This is slightly different from Schuh's original vision of an agile DBA — his focus was on how a DBA can be effective on a single team, whereas the AD method looks at the entire enterprise. The important thing is that you work in a manner appropriate to your environment.

The biggest potential change for traditional DBAs in becoming an agile DBA is that they will need to learn to work in an evolutionary manner. Modern development processes such as the Unified Process (UP) or Extreme Programming (XP) don't provide detailed requirements up front nor do they focus on detailed models (and certainly not detailed data models up front). Instead they evolve their models over time to reflect their changing understanding of the problem domain as well as the changing

requirements of their stakeholders. Some project teams may choose to work in a more serial manner, they may even choose to produce a detailed conceptual data model early in the project's life cycle, but those teams will be few and far between (although you will be expected to support them too). Agile DBAs will need to communicate the constraints imposed by legacy data sources (discussed in Chapter 8), working with application developers to understand those constraints and work appropriately.

Agile DBAs will evolve their legacy data schemas over time, applying common database refactorings (discussed in Chapter 12) as appropriate and working with new tools to evolve and migrate their data schemas over time. This is a difficult but necessary task. Agile DBAs will also need to work with application developers to model their data needs, working with UML-based artifacts such as class diagrams with some project teams and conceptual data models with other teams. Agile DBAs will work with application developers to write and test database code such as stored procedures, data-oriented code within applications that interacts with their data sources, and even aid in mapping the application schema to the data schema. Performance tuning (discussed in Chapter 15), both of the database and mappings to the database, is an important aspect of the job.

Agile DBAs commonly work with enterprise administrators, who are responsible for maintaining and evolving the corporate meta data describing the enterprise and the corporate development standards and guidelines. Agile DBAs will use this information and follow the standards and guidelines, as well as provide valuable feedback. Agile DBAs will also interact with enterprise administrators and other agile DBAs to evolve the various enterprise data sources over time, including critical meta data.

Agile DBAs also work with enterprise architects to ensure that their work fits into the overall picture and to help evolve the enterprise architecture over time.

Much of the material presented in this book is presented from the point of view of agile DBAs and application developers. I wrote it this way to remain both as consistent and as simple as possible.

Application Developers

For the sake of the agile data method an *application developer* is anyone who is actively involved with the creation and evolution of the nondata aspects of a software application (remember, any given person could take on several roles). The primary focus of an application developer is on the single system or product line that he or she is assigned to. The responsibilities of this role can include the responsibilities traditionally associated with the "traditional roles" of programmers, modelers, testers, team leads, business analysts, project managers, and deployment engineers.

As noted earlier, application developers work very closely with agile DBAs who are responsible for working on the data aspects of one or more applications. The primary customers of application developers include the potential users of their system, their managers, and the operations and support group(s) within their organization. Secondary customers include other project stakeholders such as senior management, enterprise administrators, and enterprise architects.

It is important for application developers to recognize that although their primary focus is fulfilling the current needs of direct project stakeholders, their project exists

within the larger scope of the organization. This philosophy reflects AM's (discussed in Chapter 10) principles *Software Is Your Primary Goal* and *Enabling The Next Effort Is Your Secondary Goal* — in this case part of the next effort is ensuring that your project conforms to the overall enterprise vision. Application developers are best served by recognizing that they are working on one project of many within their organization, that many projects came before theirs, that many projects will come after theirs, and that, therefore, they need to work with people in the other roles to ensure that they do the right thing.

Application developers will adopt and follow agile software development processes such as FDD, DSDM, and XP. When it comes to modeling and documentation, they are likely to enhance these processes with the principles and practices of AM. All three of these processes, being agile, implore developers to work closely with their project stakeholders. An implication is that developers are responsible for helping to educate their stakeholders, including both users and managers, in the basics of software development to help them make more informed decisions when it comes to technology.

An organization's legacy systems, including legacy data sources (discussed in Chapter 8), will constrain the efforts of application developers. These systems will often be very difficult to evolve, and if they can evolve it will often happen very slowly. Luckily, agile DBAs will be able to help application developers deal with the realities imposed upon them by legacy data sources, but they will need to work with enterprise administrators and more so with enterprise architects to ensure that their efforts reflect the long-term needs of your organization. Like agile DBAs, application developers will also need to recognize that they need to follow their organization's development practices, including the guidelines and standards supported by enterprise administrators. Application developers are expected to provide feedback regarding the standards and guidelines; everyone in the organization should do so and be prepared to work with the enterprise administrators to develop guidelines for development environments that are new to the organization.

Application developers also need to work closely with enterprise architects to ensure that their project takes advantage of existing enterprise resources and fits properly into the overall enterprise vision. The enterprise architects should be able to provide this guidance and will work with your team to architect and even build your system. Furthermore, application developers should expect to be mentored in "senior" skills such as architecture and modeling. This approach makes it easy for your team to support enterprise efforts and helps keep the enterprise architects grounded because they quickly discover whether their architecture actually works in practice.

Enterprise Administrators

An *enterprise administrator* is anyone who is actively involved in identifying, documenting, evolving, protecting, and eventually retiring corporate IT assets. These assets include corporate data, corporate development standards/guidelines, and reusable software such as components, frameworks, and services. The responsibilities of this role potentially include, but are not limited to, the responsibilities associated with traditional roles of data administrators, network administrators, reuse engineers, and software process specialists. Enterprise administrators work closely with enterprise

architects, although their primary customer teams are senior management and project teams. In many ways enterprise administrators are the "keepers of the corporate gates," supporting project teams, while at the same time guiding them to ensure that the long-term vision of the enterprise is fulfilled. An important goal is to guard and improve the quality of corporate assets, including but not limited to data. Good enterprise administrators are generalists with one or more specialties, one of which could be data administration, who understand a wide range of issues pertinent to the enterprise.

Enterprise administrators recognize that there is more to this job than data administration and will work in an evolutionary manner when supporting agile software-development teams. This is because enterprise administrators work closely with agile DBAs, and to a lesser extent application developers, who work in this manner. Enterprise administrators work with agile DBAs to ensure that their databases reflect the overall needs and direction of the enterprise. Enterprise administrators will find ways to communicate the importance of their role to agile DBAs and application developers, and the best way to do this is to focus on things that will make them more effective in their jobs — few people refuse a helping hand. Trying to impose your will through onerous processes or management edicts very likely won't work.

Enterprise administrators work with both agile DBAs and application developers to ensure that these folks understand the corporate standards and guidelines that they should follow. However, their role is to support the standards and guidelines, not enforce them. A good rule of thumb is that if you need to act as the "standards police," then you have lost the battle. Furthermore, this failure is very likely your fault because you didn't communicate the standards well, didn't gain support, or tried to enforce unrealistic guidelines. If the standards and guidelines make sense, they're written well, and they're easy to conform to, data and application developers will be willing to follow them. However, when this is not the case, when the standards and guidelines aren't appropriate or place an inordinate burden on projects, enterprise administrators should expect pushback. Yes, some individuals may chaff at following standards and guidelines but that's something that project coaches/managers will need to deal with.

When pushback occurs, an enterprise administrator works with the project team(s) to explore and address the problem. They are prepared to evolve the standards and guidelines over time to reflect lessons learned and the changing realities of the organization. One size will not fit all — your relational database naming conventions may be very different from your Java naming conventions and that's okay because those are two different environments with two different sets of priorities.

Enterprise administrators work closely with enterprise architects to communicate the constraints imposed by the current environment to the architects. More importantly, the enterprise administrators need to understand the future direction envisioned by the enterprise architects to ensure that their efforts support the long-term direction of the organization.

Enterprise Architects

An *enterprise architect* is anyone who is actively involved in the creation, evolution, and support/communication of the enterprise architecture. The architecture will often be

described as a collection of models. These models describe a wide variety of views, one of which may be data oriented, although network/hardware views, business process views, usage views, and organizational structure views (to name a few) are equally as valuable. The responsibilities of this role includes, but is not limited to, the responsibilities associated with the traditional roles of enterprise data architects, enterprise process architects, enterprise network architects, and so on.

As with the role of enterprise administrator, the role of enterprise architect has a greater scope than just that of dealing with data — instead they look at the entire enterprise picture. The enterprise architect's main job is to look into the future, to attempt to identify a direction in which the organization is going, and hence to determine how its IT infrastructure needs to evolve. Enterprise architects are naturally constrained by the current situation the organization finds itself in, its environment, and its ability to evolve. Enterprise architects work closely with enterprise administrators to ensure that they understand the current environment and to communicate their vision for the future. The primary customers of enterprise architects are the organization's senior management, including both IT management and business management, whom they work with to evolve the enterprise vision. The project teams are also primary customers because their work should reflect the overall enterprise architecture and because they provide critical feedback to that architecture.

Enterprise architects focus on a wide variety of architectural issues, data being only one of them. Their main goal is to develop and then support enterprise architectural models. It isn't sufficient for an enterprise architect to produce good models, he or she must evangelize those models, work with development teams, and educate senior management in the implications of the architecture of system-related issues in general. In addition to the CIO and CTO of your organization, your enterprise architects are likely to have the most visibility with senior management; therefore, they need to be prepared to aid senior management to make strategic decisions.

Enterprise architects work with agile DBAs and with application developers. The most important thing that enterprise architects can do is to "walk the talk" and roll up their sleeves and get actively involved with the project. This will earn the respect of the developers, dramatically increasing the chance that they'll actually understand and follow the vision of the enterprise architecture. The advantage of this approach is that it provides immediate and concrete feedback as to whether the architecture actually works and provides valuable insights for how the architecture needs to evolve.

Enterprise architects need to be prepared to work in an iterative and incremental manner. They are ill advised to try to create an all-encompassing set of enterprise models up front. Instead, create an initial, high-level architecture, and then work closely with one or more development teams to make sure that it works. AM includes a practice called *Model In Small Increments* that is based on the premise that the longer you model without receiving concrete feedback, such at that provided by an actual project, the greater the chance that your model doesn't reflect the real-world needs of your organization. Agile enterprise architects avoid ivory-tower architectures this way. An agile approach to enterprise architecture is described at the following Web page: www.agiledata.org/essays/enterpriseArchitecture.html.

Agile Software Developers

An underlying assumption of the AD method is that your organization wants to take an agile approach to software development. Agile software development reflects a shift of mindset, a new way of thinking. To succeed at the AD method, people in the four roles described earlier must have this mindset, a mindset that is characterized by the following traits:

Teamwork. Agile software developers recognize the importance of working together effectively with others and will act accordingly. They have the humility to respect and appreciate the views and abilities of others; without this humility, they are unlikely to willingly choose to collaborate with others. A critical implication is that everyone is going to have to rethink the way that they work and be willing to change for the greater good. The attitude that "my group is the center of the universe and everyone has to conform to our vision and follow our process" doesn't work well.

Common, effective processes. Agile software developers actively seek to define an overall approach that everyone agrees to. My experience is that processes imposed from the top are very likely to fail because all it takes is one group to reject the process and "go rogue." A better process-improvement strategy is to organically grow a workable software process that reflects the needs of everyone involved. Because software developers are intelligent people with valuable skills, you are likely to find that a collection of principles that everyone agrees to is often the most important part of an effective process. In the case of the AD method these are the values and principles of agile software development as well as the AD philosophies.

Co-location. Agile software developers are willing to co-locate with others as needed. You might need to give up your comfortable cubicle or office for a while to work in a shared team space, or even have someone share your office to work with you on a project. This reflects the fact that communication and collaboration are critical to your success; you are much more effective working with others than you are working alone.

Generalizing specialists. Agile software developers are generalists with one or more specialties. The implication is that everyone needs to have a wide range of skills and be willing to work with others to improve upon existing skills and to learn new ones. (Chapter 23 explores this concept in greater detail.)

Process flexibility. Agile software developers are also prepared to tailor their approach to meet the needs of the projects they are involved with. For example, a project team working on a reporting database may very well take a different approach than one working on an online application written in Java or C#. No single approach suits all situations.

Sufficient documentation. Agile software developers recognize that documentation is a necessity in their jobs, something they can be very effective at if they choose. For example, enterprise architects recognize that the goal of enterprise

modeling is to produce effective models that meet the needs of their audience, not to produce reams of documentation. They recognize that many traditional architectural efforts fail because developers are not willing to invest the time to wade through the documentation to learn the architecture. Application developers realize that system documentation is required to support future enhancement efforts and agile DBAs realize that documentation is required that describes the data sources that they support. Agile software developers will take an agile approach to documentation (discussed in Chapter 10) and produce well-written and concise documents that are just barely good enough.

Does Agile Data Solve Our Problems?

An important question to ask is whether the philosophies and suggested cultural changes discussed in this book address the problems that organizations face when it comes to the data aspects of software development. The following list shows that this in fact is the case, discussing each of the potential problems mentioned earlier in the chapter and the solution suggested by the AD method.

Different visions and priorities. Agile data implores software developers to work together and to understand and respect the viewpoints of their coworkers.

Overspecialization of roles. Agile data asks software developers to find the "sweet spot" between the extremes of being a generalist and being a specialist, ideally by becoming a generalist with one or more specialties.

Process impedance mismatch. Agile data makes it clear that enterprise and data professionals must to be prepared to work following an incremental and iterative approach, the norm for most modern development and the defacto standard for agile software development. It also makes it apparent that application developers must recognize that the existing environment, and future vision for the organization, places constraints on their efforts.

Technology impedance mismatch. Agile data requires that software developers work together closely, learning from each other as they do so. Agile DBAs have the skills to map the application schema to the data schema, to write data-oriented code, and to performance tune their work.

Ossified management. Agile data asks enterprise architects to work with senior management and educate them in the realities of modern software development. Similarly, application developers should work with and help educate all levels of management.

Organizational challenges. Agile data requires software developers to work with one another and with your project stakeholders, to respect them, and to actively strive to work together effectively.

Poor documentation. Agile data directs software developers to follow the principles of Agile Documentation (discussed in Chapter 10).

Ineffective architectural efforts. Agile data advises enterprise architects to take a multiview/model approach to architecture and to actively work on a project team to support and prove that their architecture works. The feedback from these efforts should then be reflected in future iterations of the architecture.

Ineffective development guidelines. Agile data implores enterprise administrators to write clear, effective, and applicable standards and guidelines and to be prepared to act on feedback from the development teams.

Ineffective modeling efforts. Agile data directs software developers to follow the principles and practices of the AM methodology (discussed in Chapter 10).

Summary

The heart of the agile data method is its philosophies and the changes that those philosophies imply for the way that software developers approach their jobs. The first step is to recognize that you have a problem; many organizations have a serious problem with respect to how their application developers and data professionals work together on one level, as well as how project team members work together with enterprise team members on another level. However, it isn't enough for the agile data method to merely present a collection of philosophies, it must also describe real-world, proven techniques that software developers can apply on the job. You should consider these techniques, select the ones that sound like they'll benefit you, tailor them, and apply them appropriately within your environment. Software developers can work together effectively, but they must choose to do so.